Metaproxy - User’s Guide and Reference

Copyright © 2005-2015 Index Data

COLLABORATORS

	TITLE : Metaproxy - User's Guide and Reference	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Adam Dickmeiss, Marc Cromme, and Mike Taylor	September 4, 2015	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction **1**

2 Installation **3**

2.1 Installation on Unix (from Source) . . . 3

2.1.1 Libxml2/libxslt . . . 4

2.1.2 USEMARCON (optional) . . . 4

2.1.3 YAZ/YAZ++ . . . 4

2.1.4 Boost . . . 4

2.1.5 Metaproxy . . . 5

2.2 Installation on Debian GNU/Linux . . . 5

2.3 Installation on RPM based Linux Systems . . . 6

2.4 Installation on Windows . . . 6

2.4.1 Boost . . . 6

2.4.2 Libxslt . . . 6

2.4.3 YAZ . . . 6

2.4.4 YAZ++ . . . 7

2.4.5 Metaproxy . . . 7

3 YAZ Proxy Comparison **9**

4 The Metaproxy Architecture **11**

5 Filters **13**

5.1 Introductory notes . . . 13

5.2 Overview of filter types . . . 13

5.2.1 `auth_simple` (mp::filter::AuthSimple) . . . 14

5.2.2 `backend_test` (mp::filter::Backend_test) . . . 14

5.2.3 `bounce` (mp::filter::Bounce) . . . 14
5.2.4 `cql_rpn` (mp::filter::CQLtoRPN) . . . 14
5.2.5 `frontend_net` (mp::filter::FrontendNet) . . . 14
5.2.6 `http_file` (mp::filter::HttpFile) . . . 14
5.2.7 `load_balance` (mp::filter::LoadBalance) . . . 15
5.2.8 `log` (mp::filter::Log) . . . 15
5.2.9 `multi` (mp::filter::Multi) . . . 15
5.2.10 `query_rewrite` (mp::filter::QueryRewrite) . . . 15
5.2.11 `record_transform` (mp::filter::RecordTransform) . . . 15
5.2.12 `session_shared` (mp::filter::SessionShared) . . . 15
5.2.13 `sru_z3950` (mp::filter::SRUtoZ3950) . . . 16
5.2.14 `template` (mp::filter::Template) . . . 16
5.2.15 `virt_db` (mp::filter::VirtualDB) . . . 16
5.2.16 `z3950_client` (mp::filter::Z3950Client) . . . 16
5.2.17 `zeerex_explain` (mp::filter::ZeerexExplain) . . . 16
5.3 Future directions . . . 17

6 Configuration: the Metaproxy configuration file format 19
6.1 Introductory notes . . . 19
6.2 Overview of the config file XML structure . . . 19
6.3 An example configuration . . . 20
6.4 Config file modularity . . . 21
6.5 Config file syntax checking . . . 21

7 Virtual databases and multi-database searching 23
7.1 Introductory notes . . . 23
7.2 Virtual databases with the `virt_db` filter . . . 23
7.3 Multi-database search with the `multi` filter . . . 24
7.4 What's going on? . . . 27

8 Combined SRU webservice and Z39.50 server configuration 29

9 Classes in the Metaproxy source code **31**

9.1 Introductory notes . . . 31

9.2 Individual classes . . . 31

9.2.1 `mp::FactoryFilter` (`factory_filter.cpp`) . . . 31

9.2.2 `mp::FactoryStatic` (`factory_static.cpp`) . . . 32

9.2.3 `mp::filter::Base` (`filter.cpp`) . . . 32

9.2.4 `mp::filter::AuthSimple`, `Backend_test`, etc. (`filter_auth_simple.cpp`, `filter_backend_test.cpp`, etc.) . . . 32

9.2.5 `mp::Package` (`package.cpp`) . . . 32

9.2.6 `mp::Pipe` (`pipe.cpp`) . . . 32

9.2.7 `mp::RouterChain` (`router_chain.cpp`) . . . 33

9.2.8 `mp::RouterFleXML` (`router_flexml.cpp`) . . . 33

9.2.9 `mp::Session` (`session.cpp`) . . . 33

9.2.10 `mp::ThreadPoolSocketObserver` (`thread_pool_observer.cpp`) . . . 33

9.2.11 `mp::util` (`util.cpp`) . . . 33

9.2.12 `mp::xml` (`xmlutil.cpp`) . . . 33

9.3 Other Source Files . . . 33

10 Reference **35**

10.1 metaproxy . . . 35

10.2 metaproxy-config . . . 40

10.3 auth_simple . . . 41

10.4 backend_test . . . 42

10.5 bounce . . . 43

10.6 cgi . . . 43

10.7 cql_rpn . . . 44

10.8 frontend_net . . . 46

10.9 http_client . . . 47

10.10 http_file . . . 49

10.11 http_rewrite . . . 50

10.12 limit . . . 52

10.13 load_balance . . . 54

10.14 log . . . 55

10.15 multi . . . 59

10.16 present_chunk . . . 60
10.17 query_rewrite . . . 61
10.18 record_transform . . . 62
10.19 sd_remove . . . 65
10.20 session_shared . . . 66
10.21 sort . . . 68
10.22 sru_z3950 . . . 69
10.23 template . . . 71
10.24 virt_db . . . 72
10.25 z3950_client . . . 74
10.26 zeerex_explain . . . 75
10.27 zoom . . . 76

A License **87**

B GNU General Public License **89**

B.1 Preamble . . . 89
B.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 90
B.2.1 Section 0 . . . 90
B.2.2 Section 1 . . . 90
B.2.3 Section 2 . . . 90
B.2.4 Section 3 . . . 91
B.2.5 Section 4 . . . 92
B.2.6 Section 5 . . . 92
B.2.7 Section 6 . . . 92
B.2.8 Section 7 . . . 92
B.2.9 Section 8 . . . 93
B.2.10 Section 9 . . . 93
B.2.11 Section 10 . . . 93
B.2.12 NO WARRANTY Section 11 . . . 93
B.2.13 Section 12 . . . 93
B.3 How to Apply These Terms to Your New Programs . . . 94

List of Tables

3.1 Metaproxy / YAZ Proxy comparison 10

8.1 SRU/Z39.50 Server Filter Route Configuration 30

Abstract

This manual is part of Metaproxy version 1.9.3.

Metaproxy is a universal router, proxy and encapsulated metasearcher for information retrieval protocols. It accepts, processes, interprets and redirects requests from IR clients using standard protocols such as the binary ANSI/NISO Z39.50 and the information search and retrieval web service SRU as well as functioning as a limited HTTP server.

Metaproxy is configured by an XML file which specifies how the software should function in terms of routes that the request packets can take through the proxy, each step on a route being an instantiation of a filter. Filters come in many types, one for each operation: accepting Z39.50 packets, logging, query transformation, multiplexing, etc. Further filter-types can be added as loadable modules to extend Metaproxy functionality, using the filter API.

Metaproxy is covered by the GNU General Public License version 2.

Chapter 1

Introduction

Metaproxy is a stand alone program that acts as a universal router, proxy and encapsulated metasearcher for information retrieval protocols such as Z39.50 and SRU. To clients, it acts as a server of these protocols: it can be searched, records can be retrieved from it, etc. To servers, it acts as a client: it searches in them, retrieves records from them, etc. it satisfies its clients' requests by transforming them, multiplexing them, forwarding them on to zero or more servers, merging the results, transforming them, and delivering them back to the client. In addition, it acts as a simple HTTP server; support for further protocols can be added in a modular fashion, through the creation of new filters.

```
  Anything goes in!
  Anything goes out!
  Fish, bananas, cold pyjamas,
  Mutton, beef and trout!
- attributed to Cole Porter.
```

Metaproxy is a more capable alternative to YAZ Proxy, being more powerful, flexible, configurable and extensible. Among its many advantages over the older, more pedestrian work are support for multiplexing (encapsulated metasearching), routing by database name, authentication and authorization and serving local files via HTTP. Equally significant, its modular architecture facilitites the creation of pluggable modules implementing further functionality.

This manual will describe how to install Metaproxy before giving an overview of its architecture, then discussing the key concept of a filter in some depth and giving an overview of the various filter types, then discussing the configuration file format. After this come several optional chapters which may be freely skipped: a detailed discussion of virtual databases and multi-database searching, some notes on writing extensions (additional filter types) and a high-level description of the source code. Finally comes the reference guide, which contains instructions for invoking the **metaproxy** program, and detailed information on each type of filter, including examples.

Chapter 2

Installation

Metaproxy depends on the following tools/libraries:

YAZ++ This is a C++ library based on YAZ.

Libxslt This is an XSLT processor - based on Libxml2. Both Libxml2 and Libxslt must be installed with the development components (header files, etc.) as well as the run-time libraries.

Boost The popular C++ library. Initial versions of Metaproxy was built with 1.32 but this is no longer supported. Metaproxy is known to work with Boost version 1.33 through 1.55.

In order to compile Metaproxy a modern C++ compiler is required. Boost, in particular, requires the C++ compiler to facilitate the newest features. Refer to Boost Compiler Status for more information.

We have successfully built Metaproxy using the compilers GCC and Microsoft Visual Studio.

As an option, Metaproxy may also be compiled with USEMARCON support which allows for MARC conversions for the record_transform(3mp) filter.

2.1 Installation on Unix (from Source)

Here is a quick step-by-step guide on how to compile all the tools that Metaproxy uses. Only few systems have none of the required tools binary packages. If, for example, Libxml2/libxslt are already installed as development packages use those (and omit compilation).

Note
USEMARCON is not available as a package at the moment, so Metaproxy must be built from source if that is to be used.

2.1.1 Libxml2/libxslt

Libxml2/libxslt:

```
gunzip -c libxml2-version.tar.gz|tar xf -
cd libxml2-version
./configure
make
su
make install
```

```
gunzip -c libxslt-version.tar.gz|tar xf -
cd libxslt-version
./configure
make
su
make install
```

2.1.2 USEMARCON (optional)

```
gunzip -c usemarcon317.tar.gz|tar xf -
cd usemarcon317
./configure
make
su
make install
```

2.1.3 YAZ/YAZ++

```
gunzip -c yaz-version.tar.gz|tar xf -
cd yaz-version
./configure
make
su
make install
```

```
gunzip -c yazpp-version.tar.gz|tar xf -
cd yazpp-version
./configure
make
su
make install
```

2.1.4 Boost

Metaproxy needs components thread and test from Boost.

```
gunzip -c boost-version.tar.gz|tar xf -
cd boost-version
./configure --with-libraries=thread,test,regex --with-toolset=gcc
make
su
make install
```

However, under the hood bjam is used. You can invoke that with

```
./bjam --toolset=gcc --with-thread --with-test --with-regex stage
```

Replace `stage` with `clean` / `install` to perform clean and install respectively.

Add `--prefix=DIR` to install Boost in other prefix than `/usr/local`.

2.1.5 Metaproxy

```
gunzip -c metaproxy-version.tar.gz|tar xf -
cd metaproxy-version
./configure
make
su
make install
```

You may have to tell configure where Boost is installed by supplying options `--with-boost` and `--with-boost-toolset`. The former sets the PREFIX for Boost (same as --prefix for Boost above). The latter the compiler toolset (eg. gcc34).

Pass `--help` to configure to get a list of available options.

2.2 Installation on Debian GNU/Linux

All dependencies for Metaproxy are available as Debian packages.

The procedures for Debian based systems, such as Ubuntu is probably similar

There is currently no official Debian package for YAZ++. And the official Debian package for YAZ is probably too old. But Index Data builds "new" versions of those for Debian (i386, amd64 only).

Update the `/etc/apt/sources.list` to include the Index Data repository. See YAZ' Download Debian for more information.

```
apt-get install libxslt1-dev
apt-get install libyazpp6-dev
apt-get install libboost-dev
apt-get install libboost-system-dev
apt-get install libboost-thread-dev
apt-get install libboost-test-dev
apt-get install libboost-regex-dev
```

With these packages installed, the usual configure + make procedure can be used for Metaproxy as outlined in Section 2.1.

2.3 Installation on RPM based Linux Systems

All external dependencies for Metaproxy are available as RPM packages, either from your distribution site, or from the RPMfind site.

For example, an installation of the requires Boost C++ development libraries on RedHat Fedora C4 and C5 can be done like this:

```
wget ftp://fr.rpmfind.net/wlinux/fedora/core/updates/testing/4/SRPMS/ ↩
   boost-1.33.0-3.fc4.src.rpm
sudo rpmbuild --buildroot src/ --rebuild -p fc4/boost-1.33.0-3.fc4.src. ↩
   rpm
sudo rpm -U /usr/src/redhat/RPMS/i386/boost-*rpm
```

The YAZ library is needed to compile Metaproxy, see there for more information on available RPM packages.

There is currently no official RPM package for YAZ++. See the YAZ++ pages for more information on a Unix tarball install.

With these packages installed, the usual configure + make procedure can be used for Metaproxy as outlined in Section 2.1.

2.4 Installation on Windows

Metaproxy has been tested Microsoft Visual Studio. 2013 (C 12.0).

2.4.1 Boost

For Windows, it's easiest to get the precompiled Boost package from here. Several versions of the Boost libraries may be selected when installing Boost for windows. Please choose at least the *multithreaded* (non-DLL) version because the Metaproxy makefile uses that.

For more information about installing Boost refer to the getting started pages.

2.4.2 Libxslt

Libxslt can be downloaded for Windows from here.

Libxslt also requires libxml2 to operate.

2.4.3 YAZ

YAZ can be downloaded for Windows from here.

2.4.4 YAZ++

Get YAZ++ as well. Version 1.6.0 or later is required.

YAZ++ includes NMAKE makefiles, similar to those found in the YAZ package.

2.4.5 Metaproxy

Metaproxy is shipped with NMAKE makefiles as well - similar to those found in the YAZ++/YAZ packages. Adjust this Makefile to point to the proper locations of Boost, Libxslt, Libxml2, zlib, iconv, yaz and yazpp.

DEBUG If set to 1, the software is compiled with debugging libraries (code generation is multi-threaded debug DLL). If set to 0, the software is compiled with release libraries (code generation is multi-threaded DLL).

BOOST Boost install location

BOOST_VERSION Boost version (replace . with _).

BOOST_TOOLSET Boost toolset.

LIBXSLT_DIR, LIBXML2_DIR .. Specify the locations of Libxslt, libiconv, libxml2 and libxslt.

After successful compilation you'll find `metaproxy.exe` in the `bin` directory.

Chapter 3

YAZ Proxy Comparison

The table below lists facilities either supported by either YAZ Proxy or Metaproxy.

Facility	Metaproxy	YAZ Proxy
Z39.50 server	Using filter frontend_net(3mp)	Supported
SRU server	Supported with filter sru_z3950(3mp)	Supported
Z39.50 client	Supported with filter z3950_client(3mp)	Supported
SRU client	Supported with filter zoom(3mp)	Unsupported
Connection reuse	Supported with filter `session_shared`	Supported
Connection share	Supported with filter `session_shared`	Unsupported
Result set reuse	Supported with filter `session_shared`	Within one Z39.50 session / HTTP keep-alive
Record cache	Supported by filter `session_shared`	Supported for last result set within one Z39.50/HTTP-keep alive session
Z39.50 Virtual database, i.e. select any Z39.50 target for database	Supported with filter `virt_db`	Unsupported
SRU Virtual database, i.e. select any Z39.50 target for path	Supported with filter `virt_db`, `sru_z3950`	Supported
Multi target search	Supported with filter `multi` (round-robin)	Unsupported
Retrieval and search limits	Supported using filter `limit`	Supported
Bandwidth limits	Supported using filter `limit`	Supported
Connect limits	Supported by filter `frontend_net` (connect-max)	Supported
Retrieval sanity check and conversions	Supported using filter `record_transform`	Supported
Query check	Supported by `query_rewrite` which may be check a query and throw diagnostics (errors)	Supported
Query rewrite	Supported with `query_rewrite`	Unsupported
Session invalidate for -1 hits	Unsupported	Supported
Architecture	Multi-threaded + select for networked modules such as `frontend_net`)	Single-threaded using select
Extensability	Most functionality implemented as loadable modules	Unsupported and experimental
USEMARCON	Supported with `record_transform`	Supported
Portability	Requires YAZ, YAZ++ and modern C++ compiler supporting Boost.	Requires YAZ and YAZ++. STL is not required so pretty much any C++ compiler out there should work.

Table 3.1: Metaproxy / YAZ Proxy comparison

Chapter 4

The Metaproxy Architecture

The Metaproxy architecture is based on three concepts: the *package*, the *route* and the *filter*.

Packages A package is request or response, encoded in some protocol, issued by a client, making its way through Metaproxy, send to or received from a server, or sent back to the client.

The core of a package is the protocol unit - for example, a Z39.50 Init Request or Search Response, or an SRU searchRetrieve URL or Explain Response. In addition to this core, a package also carries some extra information added and used by Metaproxy itself.

In general, packages are doctored as they pass through Metaproxy. For example, when the proxy performs authentication and authorization on a Z39.50 Init request, it removes the authentication credentials from the package so that they are not passed onto the back-end server; and when search-response packages are obtained from multiple servers, they are merged into a single unified package that makes its way back to the client.

Routes Packages make their way through routes, which can be thought of as programs that operate on the package data-type. Each incoming package initially makes its way through a default route, but may be switched to a different route based on various considerations. Routes are made up of sequences of filters (see below).

Filters Filters provide the individual instructions within a route, and effect the necessary transformations on packages. A particular configuration of Metaproxy is essentially a set of filters, described by configuration details and arranged in order in one or more routes. There are many kinds of filter - about a dozen at the time of writing with more appearing all the time - each performing a specific function and configured by different information.

The word ``filter'' is sometimes used rather loosely, in two different ways: it may be used to mean a particular *type* of filter, as when we speak of ``the auth_simple filter'' or ``the multi filter''; or it may be used to be a specific *instance* of a filter within a Metaproxy configuration. For example, a single configuration will often contain multiple instances of the `z3950_client` filter. In operational terms, of these is a separate filter. In practice, context always make it clear which sense of the word ``filter'' is being used.

Extensibility of Metaproxy is primarily through the creation of plugins that provide new filters. The filter API is small and conceptually simple, but there are many details to master. See the section below on Filters.

Since packages are created and handled by the system itself, and routes are conceptually simple, most of the remainder of this document concentrates on filters. After a brief overview of the filter types follows, along with some thoughts on possible future directions.

Chapter 5

Filters

5.1 Introductory notes

It's useful to think of Metaproxy as an interpreter providing a small number of primitives and operations, but operating on a very complex data type, namely the ``package".

A package represents a Z39.50 or SRU/W request (whether for Init, Search, Scan, etc.) together with information about where it came from. Packages are created by front-end filters such as `frontend_net` (see below), which reads them from the network; other front-end filters are possible. They then pass along a route consisting of a sequence of filters, each of which transforms the package and may also have side-effects such as generating logging. Eventually, the route will yield a response, which is sent back to the origin.

There are many kinds of filter: some that are defined statically as part of Metaproxy, and others may be provided by third parties and dynamically loaded. They all conform to the same simple API of essentially two methods: `configure()` is called at startup time, and is passed an XML DOM tree representing that part of the configuration file that pertains to this filter instance: it is expected to walk that tree extracting relevant information; and `process()` is called every time the filter has to processes a package.

While all filters provide the same API, there are different modes of functionality. Some filters are sources: they create packages (`frontend_net`); others are sinks: they consume packages and return a result (`backend_test`, `bounce`, `http_file`, `z3950_client`); the others are true filters, that read, process and pass on the packages they are fed (`auth_simple`, `log`, `multi`, `query_rewrite`, `record_transform`, `session_shared`, `sru_z3950`, `template`, `virt_db`).

5.2 Overview of filter types

We now briefly consider each of the types of filter supported by the core Metaproxy binary. This overview is intended to give a flavor of the available functionality; more detailed information about each type of filter is included below in Reference.

The filters are here named by the string that is used as the `type` attribute of a `<filter>` element in the configuration file to request them, with the name of the class that implements them in parentheses. (The classname is not needed for normal configuration and use of Metaproxy; it is useful only to developers.)

The filters are here listed in alphabetical order:

5.2.1 `auth_simple` (mp::filter::AuthSimple)

Simple authentication and authorization. The configuration specifies the name of a file that is the user register, which lists `username:password` pairs, one per line, colon separated. When a session begins, it is rejected unless username and passsword are supplied, and match a pair in the register. The configuration file may also specific the name of another file that is the target register: this lists lists `username:dbname,dbname...` sets, one per line, with multiple database names separated by commas. When a search is processed, it is rejected unless the database to be searched is one of those listed as available to the user.

5.2.2 `backend_test` (mp::filter::Backend_test)

A partial sink that provides dummy responses in the manner of the `yaz-ztest` Z39.50 server. This is useful only for testing. Seriously, you don't need this. Pretend you didn't even read this section.

5.2.3 `bounce` (mp::filter::Bounce)

A sink that swallows *all packages*, and returns them almost unprocessed. It never sends any package of any type further down the row, but sets Z39.50 packages to Z_Close, and HTTP_Request packages to HTTP_Response err code 400 packages, and adds a suitable bounce message. The bounce filter is usually added at end of each filter chain route to prevent infinite hanging of for example HTTP requests packages when only the Z39.50 client partial sink filter is found in the route.

5.2.4 `cql_rpn` (mp::filter::CQLtoRPN)

A query language transforming filter which catches Z39.50 `searchRequest` packages containing `CQL` queries, transforms those to `RPN` queries, and sends the `searchRequests` on to the next filters. It is among other things useful in a SRU context.

5.2.5 `frontend_net` (mp::filter::FrontendNet)

A source that accepts Z39.50 connections from a port specified in the configuration, reads protocol units, and feeds them into the next filter in the route. When the result is received, it is returned to the original origin.

5.2.6 `http_file` (mp::filter::HttpFile)

A partial sink which swallows only `HTTP_Request` packages, and returns the contents of files from the local filesystem in response to HTTP requests. It lets Z39.50 packages and all other forthcoming package types pass untouched. (Yes, Virginia, this does mean that Metaproxy is also a Web-server in its spare time. So far it does not contain either an email-reader or a Lisp interpreter, but that day is surely coming.)

5.2.7 `load_balance` (mp::filter::LoadBalance)

Performs load balancing for incoming Z39.50 init requests. It is used together with the `virt_db` filter, but unlike the `multi` filter it does send an entire session to only one of the virtual backends. The `load_balance` filter is assuming that all backend targets have equal content, and chooses the backend with least load cost for a new session.

Warning
This filter is experimental and yet not mature for heavy load production sites.

5.2.8 `log` (mp::filter::Log)

Writes logging information to standard output, and passes on the package unchanged. A log file name can be specified, as well as multiple different logging formats.

5.2.9 `multi` (mp::filter::Multi)

Performs multi-database searching. See the extended discussion of virtual databases and multi-database searching below.

5.2.10 `query_rewrite` (mp::filter::QueryRewrite)

Rewrites Z39.50 `Type-1` and `Type-101` (``RPN'') queries by a three-step process: the query is transliterated from Z39.50 packet structures into an XML representation; that XML representation is transformed by an XSLT stylesheet; and the resulting XML is transliterated back into the Z39.50 packet structure.

5.2.11 `record_transform` (mp::filter::RecordTransform)

This filter acts only on Z3950 present requests, and let all other types of packages and requests pass untouched. It's use is twofold: blocking Z3950 present requests, which the backend server does not understand and can not honor, and transforming the present syntax and elementset name according to the rules specified, to fetch only existing record formats, and transform them on the fly to requested record syntaxes.

5.2.12 `session_shared` (mp::filter::SessionShared)

This filter implements global sharing of result sets (i.e. between threads and therefore between clients), yielding performance improvements by clever resource pooling.

5.2.13 `sru_z3950` (mp::filter::SRUtoZ3950)

This filter transforms valid SRU GET/POST/SOAP searchRetrieve requests to Z3950 init, search, and present requests, and wraps the received hit counts and XML records into suitable SRU response messages. The `sru_z3950` filter processes also SRU GET/POST/SOAP explain requests, returning either the absolute minimum required by the standard, or a full pre-defined ZeeReX explain record. See the ZeeReX Explain standard pages and the SRU Explain pages for more information on the correct explain syntax. SRU scan requests are not supported yet.

5.2.14 `template` (mp::filter::Template)

Does nothing at all, merely passing the packet on. (Maybe it should be called `nop` or `passthrough`?) This exists not to be used, but to be copied - to become the skeleton of new filters as they are written. As with `backend_test`, this is not intended for civilians.

5.2.15 `virt_db` (mp::filter::VirtualDB)

Performs virtual database selection: based on the name of the database in the search request, a server is selected, and its address added to the request in a `VAL_PROXY` otherInfo packet. It will subsequently be used by a `z3950_client` filter. See the extended discussion of virtual databases and multi-database searching below.

5.2.16 `z3950_client` (mp::filter::Z3950Client)

A partial sink which swallows only Z39.50 packages. It performs Z39.50 searching and retrieval by proxying the packages that are passed to it. Init requests are sent to the address specified in the `VAL_PROXY` otherInfo attached to the request: this may have been specified by client, or generated by a `virt_db` filter earlier in the route. Subsequent requests are sent to the same address, which is remembered at Init time in a Session object. HTTP_Request packages and all other forthcoming package types are passed untouched.

5.2.17 `zeerex_explain` (mp::filter::ZeerexExplain)

This filter acts as a sink for Z39.50 explain requests, returning a static ZeeReX Explain XML record from the config section. All other packages are passed through. See the ZeeReX Explain standard pages for more information on the correct explain syntax.

Warning
This filter is not yet completed.

5.3 Future directions

Some other filters that do not yet exist, but which would be useful, are briefly described. These may be added in future releases (or may be created by third parties, as loadable modules).

`frontend_cli` (source) Command-line interface for generating requests.

`sru_client` (sink) SRU/GET and SRU/SOAP searching and retrieval.

`opensearch_client` (sink) A9 OpenSearch searching and retrieval.

Chapter 6

Configuration: the Metaproxy configuration file format

6.1 Introductory notes

If Metaproxy is an interpreter providing operations on packages, then its configuration file can be thought of as a program for that interpreter. Configuration is by means of a single XML file, the name of which is supplied as the sole command-line argument to the **metaproxy** program. (See Reference below for more information on invoking Metaproxy.)

6.2 Overview of the config file XML structure

All elements and attributes are in the namespace `http://indexdata.com/metaproxy`. This is most easily achieved by setting the default namespace on the top-level element, as here:

```
<metaproxy xmlns="http://indexdata.com/metaproxy" version="1.0">
```

The top-level element is <metaproxy>. This contains a <dlpath> element, a <start> element, a <filters> element and a <routes> element, in that order. <dlpath> and <filters> are optional; the other two are mandatory. All four are non-repeatable.

The <dlpath;> element contains a text element which specifies the location of filter modules. This is only needed if Metaproxy must load 3rd party filters (most filters with Metaproxy are built into the Metaproxy application).

The <start> element is empty, but carries a `route` attribute, whose value is the name of route at which to start running - analogous to the name of the start production in a formal grammar.

If present, <filters> contains zero or more <filter> elements. Each filter carries a `type` attribute which specifies what kind of filter is being defined (`frontend_net`, `log`, etc.) and contain various elements that provide suitable configuration for a filter of its type. The filter-specific elements are described in Reference. Filters defined in this part of the file must carry an `id` attribute so that they can be referenced from elsewhere.

<routes> contains one or more <route> elements, each of which must carry an `id` element. One of the routes must have the ID value that was specified as the start route in the <start> element's `route` attribute. Each route contains zero or more <filter> elements. These are of two types. They may be empty, but carry a `refid` attribute whose value is the same as the `id` of a filter previously defined in the <filters> section. Alternatively, a route within a filter may omit the `refid` attribute, but contain configuration elements similar to those used for filters defined in the <filters> section. (In other words, each filter in a route may be included either by reference or by physical inclusion.)

6.3 An example configuration

The following is a small, but complete, Metaproxy configuration file (included in the distribution as `metaproxy/etc/config1.xml`). This file defines a very simple configuration that simply proxies to whatever back-end server the client requests, but logs each request and response. This can be useful for debugging complex client-server dialogues.

```
<?xml version="1.0"?>
<metaproxy xmlns="http://indexdata.com/metaproxy" version="1.0">
  <dlpath>/usr/lib/metaproxy/modules</dlpath>
  <start route="start"/>
  <filters>
    <filter id="frontend" type="frontend_net">
      <port>@:9000</port>
    </filter>
    <filter id="backend" type="z3950_client">
    </filter>
  </filters>
  <routes>
    <route id="start">
      <filter refid="frontend"/>
      <filter type="log"/>
      <filter refid="backend"/>
      <filter type="bounce"/>
    </route>
  </routes>
</metaproxy>
```

It works by defining a single route, called `start`, which consists of a sequence of four filters. The first and last of these are included by reference: their `<filter>` elements have `refid` attributes that refer to filters defined within the prior `<filters>` section. The middle filter is included inline in the route.

The four filters in the route are as follows: first, a `frontend_net` filter accepts Z39.50 requests from any host on port 9000; then these requests are passed through a `log` filter that emits a message for each request; they are then fed into a `z3950_client` filter, which forwards all Z39.50 requests to the client-specified back-end Z39.509 server. Those Z39.50 packages are returned by the `z3950_client` filter, with the response data filled by the external Z39.50 server targeted. All non-Z39.50 packages are passed through to the `bounce` filter, which definitely bounces everything, including fish, bananas, cold pyjamas, mutton, beef and trout packages. When the response arrives, it is handed back to the `log` filter, which emits another message; and then to the `frontend_net` filter, which returns the response to the client.

6.4 Config file modularity

Metaproxy XML configuration snippets can be reused by other filters using the `XInclude` standard, as seen in the `/etc/config-sru-to-z3950.xml` example SRU configuration.

```
<filter id="sru" type="sru_z3950">
  <database name="Default">
   <xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
               href="explain.xml"/>
  </database>
</filter>
```

6.5 Config file syntax checking

The distribution contains RelaxNG Compact and XML syntax checking files, as well as XML Schema files. These are found in the distribution paths

```
xml/schema/metaproxy.rnc
xml/schema/metaproxy.rng
xml/schema/metaproxy.xsd
```

and can be used to verify or debug the XML structure of configuration files. For example, using the utility `xmllint`, syntax checking is done like this:

```
xmllint --noout --schema xml/schema/metaproxy.xsd etc/config-local.xml
xmllint --noout --relaxng xml/schema/metaproxy.rng etc/config-local.xml
```

(A recent version of `libxml2` is required, as support for XML Schemas is a relatively recent addition.)

You can of course use any other RelaxNG or XML Schema compliant tool you wish.

Chapter 7

Virtual databases and multi-database searching

7.1 Introductory notes

Two of Metaproxy's filters are concerned with multiple-database operations. Of these, `virt_db` can work alone to control the routing of searches to one of a number of servers, while `multi` can work together with `virt_db` to perform multi-database searching, merging the results into a unified result-set - ``metasearch in a box".

The interaction between these two filters is necessarily complex: it reflects the real, irreducible complexity of multi-database searching in a protocol such as Z39.50 that separates initialization from searching, and in which the database to be searched is not known at initialization time.

It's possible to use these filters without understanding the details of their functioning and the interaction between them; the next two sections of this chapter are ``HOW-TO" guides for doing just that. However, debugging complex configurations will require a deeper understanding, which the last two sections of this chapters attempt to provide.

7.2 Virtual databases with the `virt_db` filter

Working alone, the purpose of the `virt_db` filter is to route search requests to one of a selection of back-end databases. In this way, a single Z39.50 endpoint (running Metaproxy) can provide access to several different underlying services, including those that would otherwise be inaccessible due to firewalls. In many useful configurations, the back-end databases are local to the Metaproxy installation, but the software does not enforce this, and any valid Z39.50 servers may be used as back-ends.

For example, a `virt_db` filter could be set up so that searches in the virtual database ``lc" are forwarded to the Library of Congress bibliographic catalogue server, and searches in the virtual database ``marc" are forwarded to the toy database of MARC records that Index Data hosts for testing purposes. A `virt_db` configuration to make this switch would look like this:

```
<filter type="virt_db">
  <virtual>
    <database>lc</database>
    <target>lx2.loc.gov:210/LCDB_MARC8</target>
```

```
  </virtual>
  <virtual>
    <database>marc</database>
    <target>indexdata.com/marc</target>
  </virtual>
</filter>
```

As well as being useful in it own right, this filter also provides the foundation for multi-database searching.

7.3 Multi-database search with the `multi` filter

To arrange for Metaproxy to broadcast searches to multiple back-end servers, the configuration needs to include two components: a `virt_db` filter that specifies multiple `<target>` elements, and a subsequent `multi` filter. Here, for example, is a complete configuration that broadcasts searches to both the Library of Congress catalogue and Index Data's tiny testing database of MARC records:

```
<?xml version="1.0"?>
<metaproxy xmlns="http://indexdata.com/metaproxy" version="1.0">
  <start route="start"/>
  <routes>
    <route id="start">
      <filter type="frontend_net">
        <threads>10</threads>
        <port>@:9000</port>
      </filter>
      <filter type="virt_db">
        <virtual>
          <database>lc</database>
          <target>lx2.loc.gov:210/LCDB_MARC8</target>
        </virtual>
        <virtual>
          <database>marc</database>
          <target>indexdata.com/marc</target>
        </virtual>
        <virtual>
          <database>all</database>
          <target>lx2.loc.gov:210/LCDB_MARC8</target>
          <target>indexdata.com/marc</target>
        </virtual>
      </filter>
      <filter type="multi"/>
      <filter type="z3950_client">
        <timeout>30</timeout>
      </filter>
      <filter type="bounce"/>
    </route>
  </routes>
</metaproxy>
```

(Using a `virt_db` filter that specifies multiple `<target>` elements but without a subsequent `multi` filter yields surprising and undesirable results, as will be described below. Don't do that.)

Metaproxy can be invoked with this configuration as follows:

```
../src/metaproxy --config config-simple-multi.xml
```

And thereafter, Z39.50 clients can connect to the running server (on port 9000, as specified in the configuration) and search in any of the databases `lc` (the Library of Congress catalogue), `marc` (Index Data's test database of MARC records) or `all` (both of these). As an example, a session using the YAZ command-line client `yaz-client` is here included (edited for brevity and clarity):

```
$ yaz-client @:9000
Connecting...OK.
Z> base lc
Z> find computer
Search was a success.
Number of hits: 10000, setno 1
Elapsed: 5.521070
Z> base marc
Z> find computer
Search was a success.
Number of hits: 10, setno 3
Elapsed: 0.060187
Z> base all
Z> find computer
Search was a success.
Number of hits: 10010, setno 4
Elapsed: 2.237648
Z> show 1
[marc]Record type: USmarc
001    11224466
003 DLC
005 00000000000000.0
008 910710c19910701nju           00010 eng
010    $a 11224466
040    $a DLC $c DLC
050 00 $a 123-xyz
100 10 $a Jack Collins
245 10 $a How to program a computer
260 1  $a Penguin
263    $a 8710
300    $a p. cm.
Elapsed: 0.119612
Z> show 2
[VOYAGER]Record type: USmarc
001 13339105
005 20041229102447.0
008 030910s2004    caua          000 0 eng
035    $a (DLC)  2003112666
906    $a 7 $b cbc $c orignew $d 4 $e epcn $f 20 $g y-gencatlg
```

```
925 0  $a acquire $b 1 shelf copy $x policy default
955    $a pc10 2003-09-10 $a pv12 2004-06-23 to SSCD; $h sj05 2004-11-30 $e ↩
    sj05 2004-11-30 to Shelf.
010    $a   2003112666
020    $a 0761542892
040    $a DLC $c DLC $d DLC
050 00 $a MLCM 2004/03312 (G)
245 10 $a 007, everything or nothing : $b Prima's official strategy guide / ↩
    $c created by Kaizen Media Group.
246 3  $a Double-O-seven, everything or nothing
246 30 $a Prima's official strategy guide
260    $a Roseville, CA : $b Prima Games, $c c2004.
300    $a 161 p. : $b col. ill. ; $c 28 cm.
500    $a "Platforms: Nintendo GameCube, Macintosh, PC, PlayStation 2  ↩
   computer entertainment system, Xbox"--P. [4] of cover.
650  0 $a Video games.
710 2  $a Kaizen Media Group.
856 42 $3 Publisher description $u http://www.loc.gov/catdir/description/ ↩
   random052/2003112666.html
Elapsed: 0.150623
Z>
```

As can be seen, the first record in the result set is from the Index Data test database, and the second from the Library of Congress database. The result-set continues alternating records round-robin style until the point where one of the databases' records are exhausted.

This example uses only two back-end databases; more may be used. There is no limitation imposed on the number of databases that may be metasearched in this way: issues of resource usage and administrative complexity dictate the practical limits.

What happens when one of the databases doesn't respond? By default, the entire multi-database search fails, and the appropriate diagnostic is returned to the client. This is usually appropriate during development, when technicians need maximum information, but can be inconvenient in deployment, when users typically don't want to be bothered with problems of this kind and prefer just to get the records from the databases that are available. To obtain this latter behavior add an empty `<hideunavailable>` element inside the `multi` filter:

```
      <filter type="multi">
        <hideunavailable/>
      </filter>
```

Under this regime, an error is reported to the client only if *all* the databases in a multi-database search are unavailable.

7.4 What's going on?

Lark's vomit
This section goes into a level of technical detail that is probably not necessary in order to configure and use Metaproxy. It is provided only for those who like to know how things work. You should feel free to skip on to the next section if this one doesn't seem like fun.

Hold on tight - this may get a little hairy.

In the general course of things, a Z39.50 Init request may carry with it an otherInfo packet of type `VAL_PROXY`, whose value indicates the address of a Z39.50 server to which the ultimate connection is to be made. (This otherInfo packet is supported by YAZ-based Z39.50 clients and servers, but has not yet been ratified by the Maintenance Agency and so is not widely used in non-Index Data software. We're working on it.) The `VAL_PROXY` packet functions analogously to the absoluteURI-style Request-URI used with the GET method when a web browser asks a proxy to forward its request: see the Request-URI section of the HTTP 1.1 specification.

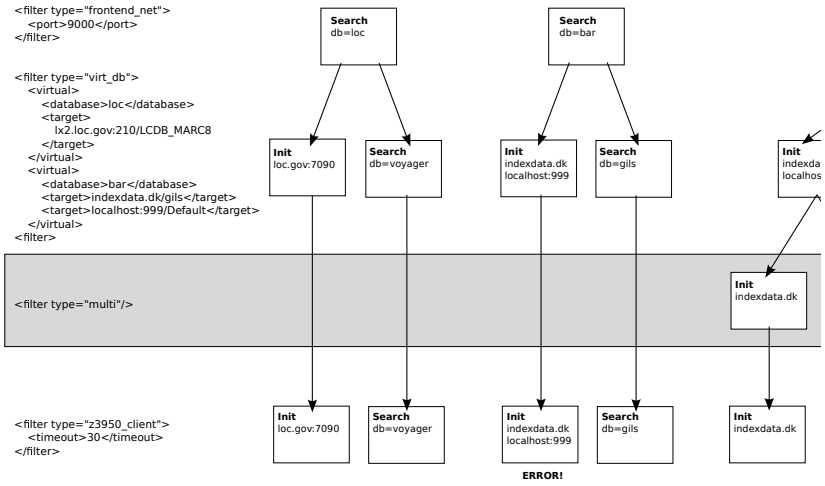
Within Metaproxy, Search requests that are part of the same session as an Init request that carries a `VAL_PROXY` otherInfo are also annotated with the same information. The role of the `virt_db` filter is to rewrite this otherInfo packet dependent on the virtual database that the client wants to search.

When Metaproxy receives a Z39.50 Init request from a client, it doesn't immediately forward that request to the back-end server. Why not? Because it doesn't know *which* back-end server to forward it to until the client sends a Search request that specifies the database that it wants to search in. Instead, it just treasures the Init request up in its heart; and, later, the first time the client does a search on one of the specified virtual databases, a connection is forged to the appropriate server and the Init request is forwarded to it. If, later in the session, the same client searches in a different virtual database, then a connection is forged to the server that hosts it, and the same cached Init request is forwarded there, too.

All of this clever Init-delaying is done by the `frontend_net` filter. The `virt_db` filter knows nothing about it; in fact, because the Init request that is received from the client doesn't get forwarded until a Search request is received, the `virt_db` filter (and the `z3950_client` filter behind it) doesn't even get invoked at Init time. The *only* thing that a `virt_db` filter ever does is rewrite the `VAL_PROXY` otherInfo in the requests that pass through it.

It is possible for a `virt_db` filter to contain multiple `<target>` elements. What does this mean? Only that the filter will add multiple `VAL_PROXY` otherInfo packets to the Search requests that pass through it. That's because the virtual DB filter is dumb, and does exactly what it's told - no more, no less. If a Search request with multiple `VAL_PROXY` otherInfo packets reaches a `z3950_client` filter, this is an error. That filter doesn't know how to deal with multiple targets, so it will either just pick one and search in it, or (better) fail with an error message.

The `multi` filter comes to the rescue! This is the only filter that knows how to deal with multiple `VAL_PROXY` otherInfo packets, and it does so by making multiple copies of the entire Search request: one for each `VAL_PROXY`. Each of these new copies is then passed down through the remaining filters in the route. (The copies are handled in parallel though the spawning of new threads.) Since the copies each have only one `VAL_PROXY` otherInfo, they can be handled by the `z3950_client` filter, which happily deals with each one individually. When the results of the individual searches come back up to the `multi` filter, it merges them into a single Search response, which is what eventually makes it back to the client.



A picture is worth a thousand words (but only five hundred on 64-bit architectures)

Chapter 8

Combined SRU webservice and Z39.50 server configuration

Metaproxy can act as SRU and web service server, which translates web service requests to ANSI/NISO Z39.50 packages and sends them off to common available targets.

A typical setup for this operation needs a filter route including the following modules:

A typical minimal example SRU server configuration file is found in the tarball distribution at `etc/config-sru-to-z3950.xml`.

Off course, any other metaproxy modules can be integrated into a SRU server solution, including, but not limited to, load balancing, multiple target querying (see Chapter 7), and complex RPN query rewrites.

Filter	Importance	Purpose
`frontend_net`	required	Accepting HTTP connections and passing them to following filters. Since this filter also accepts Z39.50 connections, the server works as SRU and Z39.50 server on the same port.
`sru_z3950`	required	Accepting SRU GET/POST/SOAP explain and searchRetrieve requests for the the configured databases. Explain requests are directly served from the static XML configuration. SearchRetrieve requests are transformed to Z39.50 search and present packages. All other HTTP and Z39.50 packages are passed unaltered.
`http_file`	optional	Serving HTTP requests from the filesystem. This is only needed if the server should serve XSLT stylesheets, static HTML files or Java Script for thin browser based clients. Z39.50 packages are passed unaltered.
`cql_rpn`	required	Usually, Z39.50 servers do not talk CQL, hence the translation of the CQL query language to RPN is mandatory in most cases. Affects only Z39.50 search packages.
`record_transform`	optional	Some Z39.50 backend targets can not present XML record syntaxes in common wanted element sets. using this filter, one can transform binary MARC records to MARCXML records, and further transform those to any needed XML schema/format by XSLT transformations. Changes only Z39.50 present packages.
`session_shared`	optional	The stateless nature of web services requires frequent re-searching of the same targets for display of paged result set records. This might be an unacceptable burden for the accessed backend Z39.50 targets, and this mosule can be added for efficient backend

Chapter 9

Classes in the Metaproxy source code

9.1 Introductory notes

Stop! Do not read this! You won't enjoy it at all. You should just skip ahead to Reference, which tells you things you really need to know, like the fact that the fabulously beautiful planet Bethselamin is now so worried about the cumulative erosion by ten billion visiting tourists a year that any net imbalance between the amount you eat and the amount you excrete whilst on the planet is surgically removed from your bodyweight when you leave: so every time you go to the lavatory it is vitally important to get a receipt.

This chapter contains documentation of the Metaproxy source code, and is of interest only to maintainers and developers. If you need to change Metaproxy's behavior or write a new filter, then you will most likely find this chapter helpful. Otherwise it's a waste of your good time. Seriously: go and watch a film or something. *This is Spinal Tap* is particularly good.

Still here? OK, let's continue.

In general, classes seem to be named big-endianly, so that `FactoryFilter` is not a filter that filters factories, but a factory that produces filters; and `FactoryStatic` is a factory for the statically registered filters (as opposed to those that are dynamically loaded).

9.2 Individual classes

The classes making up the Metaproxy application are here listed by class-name, with the names of the source files that define them in parentheses.

9.2.1 `mp::FactoryFilter` (`factory_filter.cpp`)

A factory class that exists primarily to provide the `create()` method, which takes the name of a filter class as its argument and returns a new filter of that type. To enable this, the factory must first be populated by calling `add_creator()` for static filters (this is done by the `FactoryStatic` class, see below) and `add_creator_dyn()` for filters loaded dynamically.

9.2.2 mp::FactoryStatic (factory_static.cpp)

A subclass of `FactoryFilter` which is responsible for registering all the statically defined filter types. It does this by knowing about all those filters' structures, which are listed in its constructor. Merely instantiating this class registers all the static classes. It is for the benefit of this class that `struct metaproxy_1_filter_struct` exists, and that all the filter classes provide a static object of that type.

9.2.3 mp::filter::Base (filter.cpp)

The virtual base class of all filters. The filter API is, on the surface at least, extremely simple: two methods. `configure()` is passed an XML DOM tree representing that part of the configuration file that pertains to this filter instance, and is expected to walk that tree extracting relevant information. And `process()` processes a package (see below). That surface simplicity is a bit misleading, as `process()` needs to know a lot about the `Package` class in order to do anything useful.

9.2.4 mp::filter::AuthSimple, Backend_test, etc. (filter_auth_simple.cpp, filter_backend_test.cpp, etc.)

Individual filters. Each of these is implemented by a header and a source file, named `filter_*.hpp` and `filter_*.cpp` respectively. All the header files should be pretty much identical, in that they declare the class, including a private `Rep` class and a member pointer to it, and the two public methods.

The source file for each filter needs to supply:

- A definition of the private `Rep` class.
- Some boilerplate constructors and destructors.
- A `configure()` method that uses the appropriate XML fragment.
- Most important, the `process()` method that does all the actual work.

9.2.5 mp::Package (package.cpp)

Represents a package on its way through the series of filters that make up a route. This is essentially a Z39.50 or SRU APDU together with information about where it came from, which is modified as it passes through the various filters.

9.2.6 mp::Pipe (pipe.cpp)

This class provides a compatibility layer so that we have an IPC mechanism that works the same under Unix and Windows. It's not particularly exciting.

9.2.7 mp::RouterChain (router_chain.cpp)

to be written

9.2.8 mp::RouterFleXML (router_flexml.cpp)

to be written

9.2.9 mp::Session (session.cpp)

to be written

9.2.10 mp::ThreadPoolSocketObserver (thread_pool_observer.cpp)

to be written

9.2.11 mp::util (util.cpp)

A namespace of various small utility functions and classes, collected together for convenience. Most importantly, includes the `mp::util::odr` class, a wrapper for YAZ's ODR facilities.

9.2.12 mp::xml (xmlutil.cpp)

A namespace of various XML utility functions and classes, collected together for convenience.

9.3 Other Source Files

In addition to the Metaproxy source files that define the classes described above, there are a few additional files which are briefly described here:

metaproxy_prog.cpp The main function of the **metaproxy** program.

ex_router_flexml.cpp Identical to `metaproxy_prog.cpp`: it's not clear why.

test_*.cpp Unit-tests for various modules.

Still to be described: `ex_filter_frontend_net.cpp`, `filter_dl.cpp`, `plainfile.cpp`, `tstdl.cpp`.

Chapter 10

Reference

The material in this chapter is drawn directly from the individual manual entries. In particular, the Metaproxy invocation section is available using **man metaproxy**, and the section on each individual filter is available using the name of the filter as the argument to the **man** command.

10.1 metaproxy

metaproxy — Metaproxy - server

Synopsis

`metaproxy [--help] [--version] [-v loglevel] [--config config] [-D] [-l logfile] [-m timeformat] [-p pidfile] [-t] [-u ID] [-w dir] [-X]`

DESCRIPTION

metaproxy is the Metaproxy daemon

OPTIONS

--help Display help message.

--version Display Metaproxy version.

-v *loglevel* Specify YAZ log level (all, debug, log)

--config *config* Specify the configuration.

-D Puts Metaproxy in the background after startup.

-l *logfile* Specifies YAZ log file.

-m *timeformat* Set the format of time-stamps for all logging performed via yaz_log. Refer to strftime(3) man page for the format.

-p *pidfile* Specify file which holds PID after startup.

-t Test configuration. Returns exit code 0 on success; non-zero on failure.

-u *ID* Change user ID upon startup.

-w *dir* Change working directory to *dir*.

-X Operate in debug mode.

CONFIGURATION

Metaproxy's configuration is XML based. All elements should be in namespace `http://indexdata.com/metaproxy`.

The root element must be named `metaproxy` and must specify a version. Currently the version must be `1.0`. The children elements of metaproxy are:

dlpath Specifies the path for Loadable filter modules

start Specifies the start route. Takes a `route` attribute with the name of the route.

filters Specifies all filters. Includes one ore more `filter` elements with filter-specific configuration material.

routes Specifies all routes. Includes one or more `route` elements. Each `route` in turn includes one or more filter specifications.

The configuration is described in more detail in the Metaproxy manual.

EXAMPLES

The configuration below specifies a simple Z39.50 proxy and illustrates most configuration elements of Metaproxy.

```
<?xml version="1.0"?>
<metaproxy xmlns="http://indexdata.com/metaproxy" version="1.0">
  <dlpath>/usr/local/metaproxy/filters</dlpath>
  <start route="start"/>
  <filters>
    <filter id="frontend" type="frontend_net">
      <threads>10</threads>
      <port>@:9000</port>
    </filter>
    <filter id="backend" type="z3950_client">
     <timeout>30</timeout>
     <default_target>z3950.indexdata.com</default_target>
```

```
    </filter>
  </filters>
  <routes>
    <route id="start">
      <filter refid="frontend"/>
      <filter type="log">
        <message>log</message>
      </filter>
      <filter refid="backend"/>
      <filter type="bounce"/>
    </route>
  </routes>
</metaproxy>
```

Start server with configuration in `my.xml`.

```
       metaproxy --config my.xml
```

SCHEMA

```
# Metaproxy XML config file schemas
#
#   Copyright (C) Index Data.
#
#   See the LICENSE file for details
#
#
# The RelaxNG Compact Syntax file 'metaproxy.rnc' is the master copy ↩
   .
#
# The RelaxNG XML Syntax and XML Schema are generated using 'trang':
# trang metaproxy.rnc metaproxy.rng
# trang metaproxy.rnc metaproxy.xsd
#
# Config file validation is done using 'xmllint':
# xmllint -/-relaxng metaproxy.rng ../../../etc/config1.xml
# xmllint -/-schema metaproxy.xsd config-bytarget.xml
#
# For information on RelaxNG see http://relaxng.org
# see also http://books.xmlschemata.org/relaxng/

namespace mp = "http://indexdata.com/metaproxy"

start |= metaproxy

include "filter_auth_simple.rnc"
```

```
include "filter_backend_test.rnc"
include "filter_bounce.rnc"
include "filter_cgi.rnc"
include "filter_cql_rpn.rnc"
include "filter_frontend_net.rnc"
include "filter_http_client.rnc"
include "filter_http_file.rnc"
include "filter_http_rewrite.rnc"
include "filter_http_rewrite1.rnc"
include "filter_limit.rnc"
include "filter_load_balance.rnc"
include "filter_log.rnc"
include "filter_multi.rnc"
include "filter_present_chunk.rnc"
include "filter_query_rewrite.rnc"
include "filter_record_transform.rnc"
include "filter_sd_remove.rnc"
include "filter_session_shared.rnc"
include "filter_sort.rnc"
include "filter_sru_z3950.rnc"
include "filter_virt_db.rnc"
include "filter_z3950_client.rnc"
include "filter_zoom.rnc"

any = (text | element * { attribute * { text }*, any })*

metaproxy =
  element mp:metaproxy {
    attribute version { "1.0" },
    element mp:dlpath { xsd:string }?,
    element mp:start {
            attribute route { xsd:NCName }
    },
    element mp:filters { filter+ }?,
    element mp:routes { route+ }
  }

route =
  element mp:route {
    attribute id { xsd:NCName },
    filters+
  }

filters =
  filter |
  element mp:filters {
    filters+
```

```
  }

filter =
  element mp:filter {
    filter_refid
    | filter_auth_simple
    | filter_backend_test
    | filter_bounce
    | filter_cgi
    | filter_cql_rpn
    | filter_frontend_net
    | filter_http_client
    | filter_http_file
    | filter_http_rewrite
    | filter_http_rewrite1
    | filter_limit
    | filter_load_balance
    | filter_log
    | filter_multi
    | filter_present_chunk
    | filter_query_rewrite
    | filter_record_transform
    | filter_sd_remove
    | filter_session_shared
    | filter_sort
    | filter_sru_z3950
    | filter_virt_db
    | filter_z3950_client
    | filter_zoom
  }

filter_refid = attribute refid { xsd:NCName }
```

FILES

None important.

SEE ALSO

auth_simple(3mp), backend_test(3mp), bounce(3mp), frontend_net(3mp), http_file(3mp), log(3mp), multi(3mp), query_rewrite(3mp), record_transform(3mp), session_shared(3mp), sru_z3950(3mp), template(3mp), virt_db(3m z3950_client(3mp).

The Metaproxy manual.

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.2 metaproxy-config

metaproxy-config — script to get information about the installation of Metaproxy

Synopsis

`metaproxy-config [--prefix[=DIR]] [--version] [--libs] [--lalibs] [--cflags]`

DESCRIPTION

metaproxy-config is a script that returns information that your own software should use to build software that uses Metaproxy.

OPTIONS

--prefix[=*DIR*] Returns prefix of Metaproxy or assume a different one if DIR is specified.

--version Returns version of Metaproxy.

--libs Library specification be used when linking with Metaproxy libraries.

--lalibs Returns library specification.

--cflags Returns C++ Compiler flags.

FILES

prefix`/bin/metaproxy-config`

prefix`/lib/libmetaproxy*.a`

prefix`/lib/metaproxy6/modules`

prefix`/include/metaproxy`

SEE ALSO

metaproxy(1)

The Metaproxy manual.

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.3 auth_simple

auth_simple — Metaproxy Simple Authentication And Authorization Module

DESCRIPTION

Simple authentication and authorization. The configuration specifies the name of a file that is the user register, which lists `username:password` pairs, one per line, colon separated. When a session begins, it is rejected unless username and passsword are supplied, and match a pair in the register.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_auth_simple =
  attribute type { "auth_simple" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:userRegister { xsd:string }?,
  element mp:targetRegister { xsd:string }?,
  element mp:discardUnauthorisedTargets { empty }?
```

EXAMPLES

A typical configuration looks like this:

```
    <filter type="auth_simple">
     <userRegister>../etc/example.simple-auth</userRegister>
     <targetRegister>../etc/example.target-auth</targetRegister>
     <discardUnauthorisedTargets/>
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.4 backend_test

backend_test — Metaproxy Backend Test Z39.50 Server Module

DESCRIPTION

A pseudo Z39.50 server for test purposes. Similar to yaz-ztest.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_backend_test =
  attribute type { "backend_test" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?
```

EXAMPLES

A typical configuration looks like this:

```
    <filter type="backend_test"/>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.5 bounce

bounce — Metaproxy Bouncing Package Sink Module for all kind of metaproxy packages

DESCRIPTION

A sink that swallows all packages, and returns them almost unprocessed. It never sends any package of any type further down the row, but sets Z39.50 packages to Z_Close, and HTTP_Request packages to HTTP_Response err code 400 packages, and adds a suitable bounce message. The bounce filter is added at the end of filter routes to prevent infinite hanging of yet unprocessed packages. When a package is bounced, the client connection is closed as well.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_bounce =
  attribute type { "bounce" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?
```

EXAMPLES

A typical configuration looks like this:

```
    <filter type="bounce"/>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.6 cgi

cgi — Metaproxy Package CGI Module

DESCRIPTION

CGI Common Gateway Interface module.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_cgi =
  attribute type { "cgi" },
  element mp:documentroot { xsd:string },
  element mp:env {
    attribute name { xsd:string },
    attribute value { xsd:string }
  }*,
  element mp:map {
    attribute path { xsd:string },
    attribute exec { xsd:string }
  }*
```

EXAMPLES

A typical configuration looks like this:

```
      <filter type="cgi">
         <map path="/mycgi" exec="./cgi.sh"/>
      </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.7 cql_rpn

cql_rpn — Metaproxy CQL to RPN Query Language Transforming Module

DESCRIPTION

A query language transforming filter which catches Z39.50 `searchRequest` packages containing `CQL` queries, transforms those to `RPN` queries, and sends the `searchRequests` on to the next filters.

The filter takes only one configuration parameter, namely the path of the standard `YAZ CQL`-to-`CQL` configuration file. See the YAZ manual for configuration file syntax and details.

A common and wellknown challenge is that the ZeeRex SRU Explain config file used in the `sru_z3 950` filter and the `CQL` translation configuration file used in this filter must be kept in syncronization. Syncronization can be eased by using the provided XSLT stylesheet, `xml/xslt/explain2cqlpqf txt.xsl`, which transforms from ZeeReX Explain to the later. The example configurations have been created by running:

```
    xsltproc xml/xslt/explain2cqlpqftxt.xsl etc/explain.xml > etc/cql2pqf. ↩
       txt
```

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_cql_rpn =
  attribute type { "cql_rpn" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:conversion {
    attribute file { xsd:string },
    attribute reverse { xsd:boolean }?
  }
```

EXAMPLES

A typical configuration looks like this:

```
    <filter id="cql" type="cql_rpn">
      <conversion file="etc/cql2pqf.txt"/>
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.8 frontend_net

frontend_net — Metaproxy Network Server module that accepts Z39.50 and HTTP requests

DESCRIPTION

This is a frontend module. Listens on one or more ports and sends HTTP/Z39.50 messages to other filters.

CONFIGURATION

Element `port` is a repeating element (1 or more). The text content specifies a listening port.

Element `threads` is an optional element. The text content specifies number of worker threads for the following filters to use. The default value is 5 (5 worker threads).

Element `timeout` is an optional element. The text content is treated as an integer that specifies the session timeout in seconds for a client session (using the frontend net filter). The default value is 300 (5 minutes).

Element `connect-max` is an optional element. The text content is treated as an integer that specifies maximum number of TCP sessions from the same original IP. The special value 0 means : no connection limit. This is also the default if this element is omitted.

Element `message` is an optional element. If given and non-empty logging is performed by the frontend_net filter (to the log file as given ny option -l).

Element `stat-req` is an optional element. It specifies a URL path that triggers a report to be generated by the frontend_net filter. By default this report is disabled (same as empty value). The value itself is the path and should be prefixed with a slash. For example `/fn_stat`.

SCHEMA

```
# Metaproxy XML config file schemas

namespace mp = "http://indexdata.com/metaproxy"

filter_frontend_net =
  attribute type { "frontend_net" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:threads { xsd:integer }?,
  element mp:port {
    attribute route { xsd:NCName }?,
```

```
    attribute max_recv_bytes { xsd:integer }?,
    attribute port { xsd:integer }?,
    attribute cert_fname { xsd:string }?,
   xsd:string
  }+,
  element mp:timeout { xsd:integer }?,
  element mp:connect-max { xsd:integer }?,
  element mp:http-req-max { xsd:integer }?,
  element mp:message { xsd:string }?,
  element mp:stat-req { xsd:string }?
```

EXAMPLES

A typical configuration looks like this:

```
<filter type="frontend_net">
 <threads>10</threads>
 <port>@:9000</port>
 <message>FN</message>
 <stat-req>/fn_stat</stat-req>
</filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.9 http_client

http_client — Metaproxy HTTP File Server Module

DESCRIPTION

This module implements HTTP client functionality. Filter frontend_net + http_client in combo - acts as a normal, non-transparent, proxy.

The element default-host of configuration specifies the default host in remote URL. If this is set, frontend_net + http_client acts as a transparent HTTP proxy as well.

The configuration element, proxy, is optional and enables a remote HTTP proxy to be in use.

default-host Specifies host for transparant proxy mode.

max-redirects Maximum number of HTTP redirects. Default value is 0 (HTTP redirect disabled).

proxy Specifies HTTP proxy for outgoing connections.

x-forwarded-for Is a boolean value (false, true). If true, the peer IP address as seen in frontend_net will be added to x-forwarded HTTP header.

bind_host Is a boolean value (false, true). If true, the out going TCP connection will be bound to the same as listening IP.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_http_client =
  attribute type { "http_client" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:default-host { xsd:string }?,
  element mp:max-redirects { xsd:integer }?,
  element mp:proxy { xsd:string }?,
  element mp:x-forwarded-for { xsd:boolean }?,
  element mp:bind_host { xsd:boolean }?
```

EXAMPLES

A typical configuration looks like this:

```
<filter type="http_client">
 <proxy>localhost:3128</proxy>
</filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.10 http_file

http_file — Metaproxy HTTP File Server Module

DESCRIPTION

This module enables file access via the HTTP protocol. All URLs with a given prefix are directed to a specific document root (on local file storage). The module only serves static content.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_http_file =
  attribute type { "http_file" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:mimetypes { xsd:string }?,
  element mp:area {
    element mp:documentroot { xsd:string },
    element mp:prefix { xsd:string },
    element mp:raw { xsd:boolean }?,
    element mp:passthru { xsd:boolean }?
  }*
```

EXAMPLES

A typical configuration looks like this:

```
<filter type="http_file">
 <mimetypes>/etc/mime.types</mimetypes>
 <area>
  <documentroot>/var/www/metaproxy/html</documentroot>
  <prefix>/etc</prefix>
 </area>
</filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.11 http_rewrite

http_rewrite — Module for rewriting HTTP content and headers

DESCRIPTION

The primary purpose of this module is to rewrite links (URLs) for proxying. The configuration is divided in two sections: request and response for dealing with the HTTP request and response respectively.

Each section consists of rule and content elements. Each rule must be given a name (attribute "name") and these are referred to from content elements. The content defines what rules are invoked.

Each rule consists of one or more rewrite elements. The rewrite specifies a regular expression for matching content in the attribute "from" and the corresponding attribute "from" specifies the result. The "to" result may refer to named groups in any "from" pattern already executed. For example, in the response section a rule may refer to both groups in the response already executed and all rules exectuted in the request section.

Each content section takes exactly one "type" attribute, which specifies what area is inspected for rewriting. Type may be one of "html" (for HTML content), "headers" for HTTP headers or "quoted-literal" for Java Script type of content. The content section takes one or more "within" elements. That specifies where inside the content, each rule is being exectuted. All within must have a "rule" attribtue that specifies the rule section to be invoked (rule@name as mentioned earlier).

For "html" content, the within element takes also attributes "tag" and "attr",that specifies tag and attributes to be inspected. The attr attributes takes one or more attributes (comma separated). If no "tag" is given, the rule is performed on all attributes with the name given.

For "headers" content the within element takes "header" or "reqline" + the "rule" attribute. For "header", the rule is performed on all HTTP headers with the name in header. For "reqline", the HTTP Request line is rewritten.

For "quoted-literal" content, the within element takes only a "rule" attribute and the rule is performed on all content.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"
```

```
rewrite = element mp:rewrite {
  attribute from { xsd:string },
  attribute to { xsd:string }
}

rule = element mp:rule {
  attribute name { xsd:string },
  rewrite*
}

within = element mp:within {
  attribute tag { xsd:string }?,
  attribute attr { xsd:string }?,
  attribute type { xsd:string }?,
  attribute header { xsd:string }?,
  attribute reqline { xsd:string }?,
  attribute rule { xsd:string }
}

content = element mp:content {
  attribute type { xsd:string },
  attribute mime { xsd:string }?,
  within*
}

section = (rule | content)*

filter_http_rewrite =
  attribute type { "http_rewrite" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:request {
    attribute verbose { xsd:string },
    section
  }?,
  element mp:response {
    attribute verbose { xsd:string },
    section
  }?
```

EXAMPLES

Configuration:

```
<filter type="http_rewrite">
  <request>
    <!-- save pxhost and pxpath for later -->
    <rule name="url">
      <rewrite
         from='(?&lt;proto>https?://)(?&lt;pxhost>[^ /?#]+)/(?&lt;pxpath ↩
            >[^ /]+)/(?&lt;host>[^ /]+)(?&lt;path>[^ ]*)'
         to='${proto}${host}${path}' />
      <rewrite from='(?:Host: )(.*)' to='Host: ${host}' />
    </rule>
    <content type="headers">
      <within reqline="1" rule="url"/>
    </content>
  </request>
  <response>
    <!-- rewrite "back" - using pxhost and pxpath -->
    <rule name="url">
      <rewrite
        from='(?&lt;proto>https?://)(?&lt;host>[^/?# &quot;&apos;>]+) ↩
           /(?&lt;path>[^  &quot;&apos;>]+)'
        to='${proto}${pxhost}/${pxpath}/${host}/${path}' />
    </rule>
    <content type="headers"
      <within header="link" rule="url"/>
    </content>
    <content type="html">
      <within tag="script" attr="#text" type="quoted-literal" rule="url ↩
         "/>
      <within attr="href,src" rule="url"/>
      <within attr="onclick" type="quoted-literal" rule="url"/>
    </content>
  </response>
</filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.12 limit

limit — Metaproxy Module for imposing resource limits

DESCRIPTION

This filter offers a way to limit access for a single session to a resource (target) in order to obtain a fair resource sharing.

The limit section specifies bandwidth/pdu requests limits for an active session. The filter records bandwidth/pdu requests during the last 60 seconds (1 minute). The limit may include the elements bandwidth, pdu, retrieve and search. The bandwidth measures the number of bytes transferred within the last minute. The pdu is the number of requests in the last minute. The retrieve holds the maximum records to which may be retrieved in one Present Request. The search is the maximum number of searches within the last minute.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_limit =
  attribute type { "limit" },
  element mp:limit {
    attribute bandwidth { xsd:integer }?,
    attribute pdu { xsd:integer }?,
    attribute search { xsd:integer }?,
    attribute retrieve { xsd:integer }?
  }?
```

EXAMPLES

Configuration:

```
    <filter type="limit">
     <limit bandwidth="50000" pdu="100" search="5" retrieve="50"/>
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.13 load_balance

load_balance — Metaproxy Module balancing load among multiple identical Z39.50 targets

DESCRIPTION

This filter balances load among Z39.50 targets based on statistics gathered on number of open sessions and number of open package requests.

The load balancing is depending on targets to be specified (vhosts). Most Z39.50 clients do not specify that. For this reason, this filter is mostly used as a follower to filter virt_db.

SCHEMA

```
# Metaproxy XML config file schemas
#
#   Copyright (C) Index Data.
#
#   See the LICENSE file for details
#
#
# The RelaxNG Compact Syntax file 'metaproxy.rnc' is the master copy ↩
   .
#
# The RelaxNG XML Syntax and XML Schema are generated using 'trang':
# trang metaproxy.rnc metaproxy.rng
# trang metaproxy.rnc metaproxy.xsd
#
# Config file validation is done using 'xmllint':
# xmllint -/-relaxng metaproxy.rng ../../../etc/config1.xml
# xmllint -/-schema metaproxy.xsd config-bytarget.xml
#
# For information on RelaxNG see http://relaxng.org
# see also http://books.xmlschemata.org/relaxng/

namespace mp = "http://indexdata.com/metaproxy"

filter_load_balance =
  attribute type { "load_balance" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?
```

EXAMPLES

This configuration offers one database, `Default`, which is load-balanced between two backend servers.

```
<filter type="virt_db"/>
  <database>Default</database>
  <target>host1:210/Default</target>
  <target>host2:210/Default</target>
</filter>
<filter type="load_balance"/>
```

SEE ALSO

virt_db(3mp)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.14 log

log — Metaproxy Package Logging Module

DESCRIPTION

This filter logs packages sent - and received .

Configurable values:

message Specifies a custom message for the log message.

time-format Date+time format if log is written to a custom file (see filename configuration, below), using the format of strftime(3).

Use option -m in invocation of Metaproxy command to set format if yaz_log is used (no filename given).

filename Specifies a name of log file. If this is omitted, logging is performed using the log system of YAZ (yaz_log).

category Specifies the category of messages to be logged. The category is an XML attribute and value of attribute is a boolean; `true` for enabled; `false` for disabled. The following category attributes are supported:

access One line log messages inspired by Apache access log entries. This is a brief message stating the request and response. This is enabled by default. All other categories are disabled by default. See the section ACCESS LOG.

user-access One line log messages similar to `access` but with the authenticated user on each log line.

request-apdu Z39.50 Request APDU.

response-apdu Z39.50 Response APDU.

apdu Z39.50 APDU (request and response)

request-session Request session.

response-session Response session.

session Session (request and response)

init-options Z39.50 Init Request options

line Simple one-line log message indicating the most important things from a request and response. Available from version 1.3.38 and later.

The access log

The access is is strictly one line per entry and aims for easy mangling with tools such as awk, grep, perl etc. Many values may be omitted in the packages in which case a single dash is printed instead. This is to ensure that all values have well-defined position.

The basic format and order is

time (position 1) Full time of event

Custom message (position 2) The string as given in element `message`.

IP (position 3) IP address of origin (peer)

If category `user-acesss` is used the user is written on position 3 and the IP is written on position 4.

session (position 4) Session ID. Can be used to identify a particular Z39.50 session. For HTTP this session ID only tracks the HTTP socket (kept alive). NOT to be confused the the HTTP cookie mechanism.

elapsed (position 5) Elapsed time . The elapsed time is the time between the point in time where a package was received form the client and the point where a response was received from the next filter in chain (backend eventually).

protocol (position 6) Protocol type which is one of `Z3950` or `HTTP_Request` or `HTTP_Response`.

For packages of with protocol marker `Z3950` the the access log line is followed by the APDU type + information depending on the type. The APDU type is on position 7.

initRequest Z39.50 Initialize Request with the information username, vhost, implementation ID, implementation name, implementation version.

initResponse Z39.50 Initialize Response with the information: status (OK or FAIL), implementatino ID, implementation name, implementation version.

searchRequest Z39.50 Search Request with the information: database(s), result set ID, record syntax, query.

Multiple databases are separated by a plus-sign (+). The query itself is multiple tokens. For this reason it is placed as the last information on this log entry.

searchResponse Z39.50 Search Response with the information: status (OK or FAIL), hit count, number of records returned, next result set position.

presentRequest Z39.50 Present Request with the information: result Set ID, start position, number of records requested, record syntax, record composition.

presentResponse Z39.50 Present Response with the information: status (OK, DIAG, ERROR), number of records returned, next result set position.

scanRequest Z39.50 Scan Request with the information: database(s), number of terms requested, preferred position in response, step size, start point.

start point is a multi token value in PQF notation.

scanResponse Z39.50 Scan Response with the information: status (OK, ERROR), number of entries returned, position of term, step size.

SCHEMA

```
# Metaproxy XML config file schemas
#
#   Copyright (C) Index Data.
#
#   See the LICENSE file for details
#
#
# The RelaxNG Compact Syntax file 'metaproxy.rnc' is the master copy ↩
   .
#
# The RelaxNG XML Syntax and XML Schema are generated using 'trang':
# trang metaproxy.rnc metaproxy.rng
# trang metaproxy.rnc metaproxy.xsd
#
# Config file validation is done using 'xmllint':
# xmllint -/-relaxng metaproxy.rng ../../../etc/config1.xml
# xmllint -/-schema metaproxy.xsd config-bytarget.xml
#
# For information on RelaxNG see http://relaxng.org
# see also http://books.xmlschemata.org/relaxng/

namespace mp = "http://indexdata.com/metaproxy"
```

```
filter_log =
  attribute type { "log" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:message { xsd:string }?,
  element mp:time-format { xsd:string }?,
  element mp:filename { xsd:string }?,
  element mp:category {
    attribute user-access { xsd:boolean }?,
    attribute access { xsd:boolean }?,
    attribute init-options { xsd:boolean }?,
    attribute request-session { xsd:boolean }?,
    attribute response-session { xsd:boolean }?,
    attribute session { xsd:boolean }?,
    attribute apdu { xsd:boolean }?,
    attribute request-apdu { xsd:boolean }?,
    attribute response-apdu { xsd:boolean }?,
    attribute line { xsd:boolean }?
  }?
```

EXAMPLES

A typical configuration looks like this:

```
    <filter type="log">
     <message>B</message>
     <category access="true"/>
     <filename>logs/metaproxy.log</filename>
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.15 multi

multi — Metaproxy Package Multiplexer Module

DESCRIPTION

This filter multiplexes packages.

The multi filter consists of zero or more `<target>` elements. If a target matches a given target specified as CDATA in the target element, the multi filter will route traffic to the route given by the route attribute. The target element may also apply credentials to be sent to the target. This is given by the `auth` attribute.

A target element is not required for multi-plexing to work. It merely serves as a way to route differently.

If an empty `<hideunavailable>` element is placed inside the `multi` filter, then unavailable databases are not reported to the client, but simply ignored (unless every one of the databases is unavailable).

If an empty `<hideerrors>` element is placed inside the `multi` filter, then databases that reports diagnostics are not reported back to the client, but simply ignored (unless every one of the databases report diagnostics).

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_multi =
  attribute type { "multi" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:target {
    attribute route { xsd:string }?,
    attribute auth { xsd:string }?,
    xsd:string
  }*,
  element mp:hideunavailable { empty }?,
  element mp:hideerrors { empty }?,
  element mp:mergetype { xsd:string }?
```

EXAMPLES

A typical configuration looks like this:

```
    <filter type="multi">
     <target route="route1">lx2.loc.gov:210/LCDB_MARC8</target>
     <target route="route2">z3950.indexdata.com/gils</target>
     <target route="route3" auth="myuser/mypass">localhost:9999</target>
     <target route="other">*</target>
    </filter>
```

SEE ALSO

metaproxy(1)

virt_db(3mp)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.16 present_chunk

present_chunk — Splits Z39.50 Present Request into chunks

DESCRIPTION

This module converts a present requests to one or more present requests (chunks). Some Z39.50 server software may crash or large present requests (number of records requested). This module tries to fix that. It takes a "chunk" element in configuration that specifies the maximum number of records to fetch in each chunk.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_present_chunk =
  attribute type { "present_chunk" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:chunk { xsd:integer }?
```

EXAMPLES

Configuration:

```
    <filter type="present_chunk">
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.17 query_rewrite

query_rewrite — Metaproxy RPN Query Rewrite Module

DESCRIPTION

This module allows Z39.50 Type-1 queries to be arbitrarily rewritten using an XSLT stylesheet to specify the rewrite rules. This can be useful for several purposes, including the provision of index aliases (e.g. BIB-1 access-point 1, personal name, rewritten to access-point 1003, author); and protecting fragile Z39.50 server from attribute combinations that cause them problems.

The Type-1 query is translated into an XML representation, transformed by an XSLT stylesheet whose path is specified in the filter configuration's <xslt> element, then translated back into a Type-1 query.

Describe the XML representation.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_query_rewrite =
  attribute type { "query_rewrite" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:xslt {
        attribute stylesheet { xsd:string }
```

```
}?,
element mp:charset {
      attribute from { xsd:string }?,
      attribute to { xsd:string }?
}?
```

EXAMPLES

A typical configuration looks like this:

```
<filter type="query_rewrite">
 <xslt stylesheet="pqf2pqf.xsl">
</filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.18 record_transform

record_transform — Metaproxy Module that performs record transformations

DESCRIPTION

This filter acts on Z39.50 present requests and Z39.50 search requests, and let all other types of packages and requests pass untouched. Its use is twofold: blocking Z39.50 present/search requests that the backend server does not understand or can not honor, and transforming the present syntax and elementset name according to the rules specified, to fetch only existing record formats, and transform them on the fly to requested record syntaxes.

The allowed record present syntax and element name are described in multiple `<retrieval>` elements inside the `<retrievalinfo>` element. The `syntax` attribute is mandatory, but the `name` attribute may be omitted, in which case any element name is accepted. An additional `identifier` attribute can be added to explicitly describe the Z39.50 identifier string.

The `<retrieval>` element and the content is described in the Retrieval Facility section of the YAZ manual.

From Metaproxy version 1.3.26 and onwards the backend conversion ma also use USEMARCON. If USEMARCON is enabled, the `backend` may include a `usemarcon` element with two attributes: `stage1` and `stage2` that points to conversion files as interpreted by USEMARCON. One or both must be given.

SCHEMA

Schema is in two parts.. One for the filter itself and one for the retrieval info.

```
  # Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

include "retrievalinfo.rnc"

filter_record_transform =
  attribute type { "record_transform" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  retrievalinfo


  # Schema for YAZ retrieval info and USEMARCON extension

namespace y = "http://indexdata.com/yaz"

start |= retrievalinfo

marc = element y:marc {
  attribute inputformat { xsd:string },
  attribute outputformat { xsd:string },
  attribute inputcharset { xsd:string },
  attribute outputcharset { xsd:string }?,
  attribute leaderspec { xsd:string }?
}

xslt = element y:xslt {
  attribute stylesheet { xsd:string },
  element y:param {
  attribute name {xsd:string},
  attribute value {xsd:string}
  }*
}

usemarcon = element y:usemarcon {
  attribute stage1 { xsd:string }?,
  attribute stage2 { xsd:string }?
}
```

```
retrievalinfo =
  element y:retrievalinfo {
    attribute version { "1.0" },
    element y:retrieval {
      attribute syntax { xsd:string },
      attribute name { xsd:string }?,
      attribute identifier { xsd:string }?,
      element y:backend {
      attribute syntax { xsd:string },
      attribute name { xsd:string }?,
      (marc | xslt | usemarcon)*
      }?
    }+
  }
```

EXAMPLES

A typical configuration looks like this:

```
    <filter type="record_transform">
      <retrievalinfo  xmlns="http://indexdata.com/yaz" version="1.0">
        <retrieval syntax="xml" name="dc"
                   identifier="info:srw/schema/1/dc-v1.1">
          <backend syntax="usmarc" name="F">
            <marc inputformat="marc" outputformat="marcxml"
                  inputcharset="marc-8"/>
            <xslt stylesheet="../xml/xslt/MARC21slim2DC.xsl"/>
          </backend>
        </retrieval>
        <retrieval syntax="opac"/>
        <retrieval syntax="xml" name="opac">
          <backend syntax="opac" name="F">
          <marc inputformat="marc" outputformat="marcxml"
                inputcharset="marc-8"/>
          </backend>
        </retrieval>
        <retrieval syntax="usmarc">
          <backend syntax="usmarc" name="F">
          </backend>
        </retrieval>
        <retrieval syntax="usmarc" name="C">
    <backend syntax="unimarc" name="F">
            <usemarcon stage1="/etc/usemarcon/uni2us/uni2us.ini"/>
          </backend>
        </retrieval>
      </retrievalinfo>
```

```
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.19 sd_remove

sd_remove — Removes Surrogate Diagnostics

DESCRIPTION

This filter removes surrogate-diagnostics from Z39.50 records. It replaces the surrogate diagnostics records with SUTRS records. This module has no general use. It was only implemented to avoid a particualr Z39.50 from crashing.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_sd_remove =
  attribute type { "sd_remove" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?
```

EXAMPLES

Configuration:

```
    <filter type="sd_remove">
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.20 session_shared

session_shared — Metaproxy Module for sharing system ressorces between threads

DESCRIPTION

This filter implements global sharing of result sets (i.e. between threads and therefore between clients), yielding performance improvements especially when incoming requests are from a stateless environment such as a web-server, in which the client process representing a session might be any one of many. It performs the following actions:

- Reduce the number of backend server sessions.
- Reduce the number of initializations with backend servers.
- Optimize the use of result-sets.

Configurable values:

Session TTL When a backend session is idle for more than this amount of time, given in seconds, it will be closed. Default value is 90 seconds.

Session max Specifiers maximum number of sessions to any particular target. If this number (limit) is reached, the session_shared module will re-use result sets even if TTL is not met. Failing that the session_shared will return a diagnostic. Default value is 100 sessions.

Result-Set TTL When a backend session result-set is not in use for more than this amount of time, given in seconds, it will be deleted/reused. Default value is 30 seconds.

Result-Set max This specifies the maximum number of result-sets in use by a backend. The number only applies to targets/servers with named result sets. Targets that do not support named result sets may only have one active result set. Default value is 10.

Result-Set restart Boolean which specifies whether session_shared should to try to recover a failed search. If a search results in diagnostic 2: temporary system error, or a negative hit count, the search will be performed once again on another or new Z39.50 session. Default value is true (enabled).

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_session_shared =
  attribute type { "session_shared" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:resultset {
    attribute max { xsd:integer }?,
    attribute ttl { xsd:integer }?,
    attribute optimizesearch { xsd:boolean }?,
    attribute restart { xsd:boolean }?
  }?,
  element mp:session {
    attribute ttl { xsd:integer }?,
    attribute max { xsd:integer }?
  }?,
  element mp:init {
    attribute preferred-message-size { xsd:integer }?,
    attribute maximum-record-size { xsd:integer }?
  }?
```

EXAMPLES

Configuration:

```
    <filter type="session_shared">
     <resultset ttl="10" max="3" restart="true"/>
     <session ttl="30" max="100"/>
    </filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.21 sort

sort — Metaproxy Z39.50 Sort Module

DESCRIPTION

This filter performs sorting of Z39.50 result sets. The sorting criteria is selected via an X-Path expression. Only XML records are supported. The sorting is done only for the first present request following a search. The number of records to prefetch can is configurable. For example, if a client asks initially for 10 records this module may extend that and fetch more records and only return the results in the 10 record window - after sorting.

The configuration is given as attribute inside element `sort`. This element must occur exactly once. Future versions of the sort module may include multiple sort elements. The attributes within sort are:

xpath Specifies the X-Path expression that picks the sorting data from the record.

namespaces Allows one or more namespaces to be declared with a user-defined prefix. Each prefix may be referred to within the xpath expression.

prefetch Number of records to prefetch.

ascending Is a boolean value (false, true). If true, the sort module will sort ascending. If false, the sort module will sort descending. If omitted, the sort order will be ascending.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_sort =
  attribute type { "sort" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:sort {
    attribute prefetch { xsd:integer }?,
    attribute xpath { xsd:string },
    attribute namespaces { xsd:string }?,
    attribute ascending { xsd:boolean }?,
    attribute debug { xsd:boolean }?
  }
```

EXAMPLES

For example, to sort MARCXML records on title, one could use:

```
<filter type="sort">
  <sort
    xpath="/marc:record/marc:datafield[@tag='245']/marc:subfield[@code='a ↩
       ']"
    namespaces="marc=http://www.loc.gov/MARC21/slim"
    prefetch="5"
    ascending="true"
    debug="true"
   />
</filter>
```

SEE ALSO

metaproxy(1)

record_transform(3mp)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.22 sru_z3950

sru_z3950 — Metaproxy Module transforming SRU web service requests to Z39.50 Metaproxy packages

DESCRIPTION

The `sru_z3950` Metaproxy filter transforms valid SRU GET/POST/SOAP requests to Z39.50 requests, and wraps the received hit counts and XML records into suitable SRU response messages.

Multiple database elements defining the names of the accepted databases are allowed in the configuration file. Each of them must contain their own explain record, or must be empty. Notice that explain records come in SRU and Z39.50 flavors, and this filter requires the SRU version. See the ZeeReX Explain standard pages and the SRU Explain pages for more information.

Optionally the default stylesheet may be specified. If the client do not specify a stylesheet, the CDATA of element `stylesheet` is used.

All Z39.50 packages and all HTTP packages that do not resolve to one configured database name are passed unaltered to the next filters on the route.

The SRU `explain` operation is supported, returning either the absolute minimum required by the standard, or a full pre-defined ZeeReX explain record.

It supports the SRU `searchRetrieve` operation, which is transformed into successive Z39.50 `init`, `search` and `present` requests.

The SRU `scan` operation is not supported.

This filter does not handle CQL-to-PQF translations. In case that the backends do not understand CQL, you need to append the `cql_pqf` metaproxy filter.

This module supports the following SRU extra parameters:

x-target Specifies backend Z39.50 target.

x-max-sockets Specifies maximum number of sockets to use for a Z39.50 backend client (for one given target host/db).

x-session-id Allow a user-defined session ID to be attached to filter log that follows sru_z3950. The ID is present in the log files and not available to the SRU webservice. In order to log material out via SRU, the x-log-enable may be used instead.

x-log-enable Controls whether log is to be collected for filters that sru_z3950. Log data is extra response data's log element. A value of 1 enables logging; other value disables logging (default).

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_sru_z3950 =
  attribute type { "sru_z3950" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:stylesheet { xsd:string }?,
  element mp:database {
     attribute name { xsd:NCName },
     any
  }*
```

EXAMPLES

A typical configuration looks like this:

```
<filter type="sru_z3950">
  <stylesheet>/my.xsl</stylesheet>
  <database name="Default">
    <explain xmlns="http://explain.z3950.org/dtd/2.0/">
      ...
    </explain>
  </database>
  <database name="Dummy">
</filter>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.23 template

template — Metaproxy Template Module That Does Nothing

DESCRIPTION

This module does nothing at all, simply passing packages through untouched. It exists not to be instantiated, but to be copied by programmers creating new filters.

EXAMPLES

A typical configuration looks like this:

```
<filter type="template"/>
```

SEE ALSO

metaproxy(1)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.24 virt_db

virt_db — Metaproxy Virtual Databases Module

DESCRIPTION

This filter allows one Z39.50 database to be me mapped to another target; or even multiple targets.

The configuration of virt_db consists of zero or more `<virtual>` elements each describing the Z39.50 virtual database recognized. The name of the database is the text content of the `<database>` element which should be first element inside the virtual section.

For Metaproxy 1.0.20 and later, the database is treated as a glob pattern. This allows operators * (any number of any character) and ? (any single character). The virtual sections are inspected in the order given. The first matching virtual database is used.

Following that is one or more `<target>` elements with the identifier of each target that the virtual database maps to. If a database is given for a target (following a slash), that database name is used as Z39.50 database for this target. If the database is omitted for the target, the original (virtual) database is used as Z39.50 database.

If multiple targets are given that has special meaning depending on the filter following virt_db. If the following filter is load_balance, then the load_balance filter will load balance between the targets given (assuming they serve same content). If the following filter is multi, then results will be merged from all targets instead.

The z3950_filter, on the other hand, does not support multiple targets.

For `<virtual>` a route may be given as an attribute. This will make Metaproxy route traffic to the route given. Note that virtual databases may not be combined if all databases do not result in the same routing.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_virt_db =
  attribute type { "virt_db" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:pass-vhosts { xsd:boolean }?,
  element mp:virtual {
    attribute route { xsd:NCName }?,
```

```
    element mp:database { xsd:string },
    element mp:target { xsd:string }+
  }*
```

EXAMPLES

Consider this virt_db configuration:

```
  <filter type="virt_db">
    <virtual>
      <database>db1</database>
      <target>localhost:9999/Default</target>
    </virtual>
    <virtual>
      <database>db2</database>
      <target>z3950.indexdata.com/gils</target>
    </virtual>
    <virtual>
      <database>combined</database>
      <target>z3950.indexdata.com/gils</target>
      <target>localhost:9999/Default</target>
    </virtual>
    <virtual route="special">
      <database>db3</database>
      <target>z3950.indexdata.com/special</target>
    </virtual>
    <virtual>
      <database>*</database><!-- default -->
      <target>localhost:9999</target><-- database not altered -->
    </virtual>
  </filter>
```

This will offer 4 databases, `db1`, `db2`, `combined` and `db3`. If a Z39.50 specifies `db1` and `db2` that will have same effect as specifying the single database `combined`.

Since `db3` routes differently from the other databases, this database may not be combined with the others.

SEE ALSO

metaproxy(1)

multi(3mp) load_balance(3mp)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.25 z3950_client

z3950_client — Metaproxy Z39.50 Backend Client Module

DESCRIPTION

This backend filter is a Z39.50 client. This modules proxies all Z39.50 packages to a target. HTTP packages are ignored. The address of the backend target (host) can be given as part of the Initialize Request (Virtual host) or the default target may be specified in the configuration.

connect-timeout Specifies how long the client will wait for TCP connect to complete before giving up. Default value is 30 seconds.

init-timeout Specifies how long the client will wait for Z39.50 Init response before giving up. Default value is 10 seconds.

timeout Specifies how long the client will wait for any request other than Init before giving up. Default value is 30 seconds.

default_target Specifies the target (host) for the Z39.50 server to be used if the Init Request does not indicate otherwise.

force_target Specifies the target (host) for the Z39.50 server to be used always (regardless of Init Request vhost).

force_close Is a boolean value (false, true). If true, the Z39.50 client will terminate Z39.50 sessions with a close APDU followed by a socket close. If false (default), the Z39.50 client will be transparent and only send a close if the peer client does it too.

client_ip Is a boolean value (false, true). If true, the Z39.50 client will as part of the Init Request include Client-IP information (the Z39.50 equivalent of HTTP X-Forwarded-To information). By default this is false (not included).

charset If set, holds Z39.50 negotiation charset (encoding) that is sent via the Initialize Request. If Initalize Request already contains negotaition information it will be left un-modified.

bind_host Is a boolean value (false, true). If true, the out going TCP connection will be bound to the same as listening IP.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_z3950_client =
  attribute type { "z3950_client" },
  attribute id { xsd:NCName }?,
```

```
attribute name { xsd:NCName }?,
element mp:connect-timeout { xsd:integer }?,
element mp:init-timeout { xsd:integer }?,
element mp:timeout { xsd:integer }?,
element mp:default_target { xsd:string }?,
element mp:force_target { xsd:string }?,
element mp:force_close { xsd:boolean }?,
element mp:client_ip { xsd:boolean }?,
element mp:charset { xsd:string }?,
element mp:bind_host { xsd:boolean }?
```

EXAMPLES

A typical configuration looks like this:

```
<filter type="z3950_client">
 <timeout>30</timeout>
 <default_target>z3950.indexdata.com</default_target>
</filter>
```

SEE ALSO

metaproxy(1)

backend_test(3mp)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.26 zeerex_explain

zeerex_explain — Metaproxy Z39.50 ZeeReX Explain Module

DESCRIPTION

The `zeerex_explain` Metaproxy filter answers valid Z30.50 explain requests, returning a static ZeeReX Explain XML record from the config section. All other packages are passed through.

Multiple database elements defining the names of the accepted databases are allowed in the configuration file. Each of them must contain their own explain record. Notice that explain records come in SRU and

Z39.50 flavours, and this filter requires the Z39.50 version. See the ZeeReX Explain standard pages and the SRU Explain pages for more information.

Warning
This filter is not yet completed.

EXAMPLES

A typical configuration looks like this:

```
<filter type="zeerex_explain">
  <database name="Default">
    <explain xmlns="http://explain.z3950.org/dtd/2.0/">
      ...
    </explain>
  </database>
</filter>
```

SEE ALSO

metaproxy(1)

ZeeReX Explain

COPYRIGHT

Copyright (C) 2005-2015 Index Data

10.27 zoom

zoom — Metaproxy ZOOM Module

DESCRIPTION

This filter implements a generic client based on ZOOM of YAZ. The client implements the protocols that ZOOM C does: Z39.50, SRU (GET, POST, SOAP) and Solr .

This filter only deals with Z39.50 on input. The following services are supported: init, search, present and close. The backend target is selected based on the database as part search and *not* as part of init.

This filter is an alternative to the z3950_client filter but also shares properties of the virt_db - in that the target is selected for a specific database

The ZOOM filter relies on a target profile description, which is XML based. It picks the profile for a given database from a web service or it may be locally given for each unique database (AKA virtual database in virt_db). Target profiles are directly and indrectly given as part of the `torus` element in the configuration.

CONFIGURATION

The configuration consists of six parts: `torus`, `fieldmap`, `cclmap`, `contentProxy`, `log` and `zoom`.

torus

The `torus` element specifies target profiles and takes the following content:

attribute `url` URL of Web service to be used when fetch target profiles from a remote service (Torus normally).

The sequence `%query` is replaced with a CQL query for the Torus search.

The special sequence `%realm` is replaced by value of attribute `realm` or by realm DATABASE argument.

The special sequence `%db` is replaced with a single database while searching. Note that this sequence is no longer needed, because the `%query` can already query for a single database by using CQL query `udb==...`.

attribute `content_url` URL of Web service to be used to fetch target profile for a given database (udb) of type content. Semantics otherwise like `url` attribute above.

attribute `auth_url` URL of Web service to be used to for auth/IP lookup. If this defined, all access is granted or denied as part of Z39.50 Init by the ZOOM module and the use of database parameters realm and torus_url is not allowed. If this setting is not defined, all access is allowed and realm and/or torus_url may be used.

attribute `auth_hostname` Limits IP lookup to a given logical hostname.

attribute `realm` The default realm value. Used for %realm in URL, unless specified in DATABASE parameter.

attribute `proxy` HTTP proxy to bse used for fetching target profiles.

attribute `xsldir` Directory that is searched for XSL stylesheets. Stylesheets are specified in the target profile by the `transform` element.

attribute `element_transform` Specifies the element that triggers retrieval and transform using the parameters elementSet, recordEncoding, requestSyntax, transform from the target profile. Default value is "pz2", due to the fact that for historical reasons the common format is that used in Pazpar2.

attribute `element_raw` Specifies an element that triggers retrieval using the parameters elementSet, recordEncoding, requestSyntax from the target profile. Same actions as for element_transform, but without the XSL transform. Useful for debugging. The default value is "raw".

attribute `explain_xsl` Specifies a stylesheet that converts one or more Torus records to ZeeExplain records. The content of recordData is assumed to be holding each Explain record.

attribute `record_xsl` Specifies a stylesheet that converts retrieval records after transform/literal operations.

When Metaproxy creates a content proxy session, the XSL parameter `cproxyhost` is passed to the transform.

element `records` Local target profiles. This element may includes zero or more `record` elements (one per target profile). See section TARGET PROFILE.

fieldmap

The `fieldmap` may be specified zero or more times and specifies the map from CQL fields to CCL fields and takes the following content:

attribute `cql` CQL field that we are mapping "from".

attribute `ccl` CCL field that we are mapping "to".

cclmap

The third part of the configuration consists of zero or more `cclmap` elements that specifies *base* CCL profile to be used for all targets. This configuration, thus, will be combined with cclmap-definitions from the target profile.

contentProxy

The `contentProxy` element controls content proxy'in. This section is optional and must only be defined if content proxy'ing is enabled.

attribute `config_file` Specifies the file that configures the cf-proxy system. Metaproxy uses setting `sessiondir` and `proxyhostname` from that file to configure name of proxy host and directory of parameter files for the cf-proxy.

attribute `server` Specifies the content proxy host. The host is of the form host[:port]. That is without a method (such as HTTP) and optional port number.

> **Note**
> This setting is deprecated. Use the config_file (above) to inform about the proxy server.

attribute `tmp_file` Specifies a filename of a session file for content proxy'ing. The file should be an absolute filename that includes `XXXXXX` which is replaced by a unique filename using the mkstemp(3) system call. The default value of this setting is `/tmp/cf.XXXXXX.p`.

> **Note**
> This setting is deprecated. Use the config_file (above) to inform about the session file area.

log

The `log` element controls logging for the ZOOM filter.

attribute `apdu` If the value of apdu is "true", then protocol packages (APDUs and HTTP packages) from the ZOOM filter will be logged to the yaz_log system. A value of "false" will not perform logging of protocol packages (the default behavior).

zoom

The `zoom` element controls settings for the ZOOM.

attribute `timeout` Is an integer that specifies, in seconds, how long an operation may take before ZOOM gives up. Default value is 40.

attribute `proxy_timeout` Is an integer that specifies, in seconds, how long an operation a proxy check will wait before giving up. Default value is 1.

QUERY HANDLING

The ZOOM filter accepts three query types: RPN(Type-1), CCL and CQL.

Queries are converted in two separate steps. In the first step the input query is converted to RPN/Type-1. This is always the common internal format between step 1 and step 2. In step 2 the query is converted to the native query type of the target.

Step 1: for RPN, the query is passed un-modified to the target.

Step 1: for CCL, the query is converted to RPN via cclmap elements part of the target profile as well as base CCL maps.

Step 1: For CQL, the query is converted to CCL. The mappings of CQL fields to CCL fields are handled by fieldmap elements as part of the target profile. The resulting query, CCL, is the converted to RPN using the schema mentioned earlier (via `cclmap`).

Step 2: If the target is Z39.50-based, it is passed verbatim (RPN). If the target is SRU-based, the RPN will be converted to CQL. If the target is Solr-based, the RPN will be converted to Solr's query type.

SORTING

The ZOOM module actively handle CQL sorting - using the SORTBY parameter which was introduced in SRU version 1.2. The conversion from SORTBY clause to native sort for some target is driven by the two parameters: sortStrategy and sortmap_field.

If a sort field that does not have an equivalent `sortmap_`-mapping is passed un-modified through the conversion. It doesn't throw a diagnostic.

TARGET PROFILE

The ZOOM module is driven by a number of settings that specifies how to handle each target. Note that unknown elements are silently *ignored*.

The elements, in alphabetical order, are:

authentication Authentication parameters to be sent to the target. For Z39.50 targets, this will be sent as part of the Init Request. Authentication consists of two components: username and password, separated by a slash.

If this value is omitted or empty no authentication information is sent.

authenticationMode Specifies how authentication parameters are passed to server for SRU. Possible values are: `url` and `basic`. For the url mode username and password are carried in URL arguments x-username and x-password. For the basic mode, HTTP basic authentication is used. The settings only takes effect if authentication is set.

If this value is omitted HTTP basic authencation is used.

cclmap_*field* This value specifies CCL field (qualifier) definition for some field. For Z39.50 targets this most likely will specify the mapping to a numeric use attribute + a structure attribute. For SRU targets, the use attribute should be string based, in order to make the RPN to CQL conversion work properly (step 2).

cfAuth When cfAuth is defined, its value will be used as authentication to backend target and authentication setting will be specified as part of a database. This is like a "proxy" for authentication and is used for Connector Framework based targets.

cfProxy Specifies HTTP proxy for the target in the form *host*:*port*.

cfSubDB Specifies sub database for a Connector Framework based target.

contentAuthentication Specifies authentication info to be passed to a content connector. This is only used if content-user and content-password are omitted.

contentConnector Specifies a database for content-based proxy'ing.

elementSet Specifies the elementSet to be sent to the target if record transform is enabled (not to be confused' with the record_transform module). The record transform is enabled only if the client uses record syntax = XML and a element set determined by the `element_transform` / `element_raw` from the configuration. By default that is the element sets `pz2` and `raw`. If record transform is not enabled, this setting is not used and the element set specified by the client is passed verbatim.

literalTransform Specifies a XSL stylesheet to be used if record transform is anabled; see description of elementSet. The XSL transform is only used if the element set is set to the value of `element_transform` in the configuration.

The value of literalTransform is the XSL - string encoded.

piggyback A value of 1/true is a hint to the ZOOM module that this Z39.50 target supports piggyback searches, ie Search Response with records. Any other value (false) will prevent the ZOOM module to make use of piggyback (all records part of Present Response).

queryEncoding If this value is defined, all queries will be converted to this encoding. This should be used for all Z39.50 targets that do not use UTF-8 for query terms.

recordEncoding Specifies the character encoding of records that are returned by the target. This is primarily used for targets were records are not UTF-8 encoded already. This setting is only used if the record transform is enabled (see description of elementSet).

requestSyntax Specifies the record syntax to be specified for the target if record transform is enabled; see description of elementSet. If record transform is not enabled, the record syntax of the client is passed verbatim to the target.

sortmap_*field* This value the native field for a target. The form of the value is given by sortStrategy.

sortStrategy Specifies sort strategy for a target. One of: `z3950`, `type7`, `cql`, `sru11` or `embed`. The `embed` chooses type-7 or CQL sortby depending on whether Type-1 or CQL is actually sent to the target.

sru If this setting is set, it specifies that the target is web service based and must be one of : `get`, `post`, `soap` or `solr`.

sruVersion Specifies the SRU version to use. It unset, version 1.2 will be used. Some servers do not support this version, in which case version 1.1 or even 1.0 could be set it.

transform Specifies a XSL stylesheet filename to be used if record transform is anabled; see description of elementSet. The XSL transform is only used if the element set is set to the value of `element_transform` in the configuration.

udb This value is required and specifies the unique database for this profile . All target profiles should hold a unique database.

urlRecipe The value of this field is a string that generates a dynamic link based on record content. If the resulting string is non-zero in length a new field, `metadata` with attribute `type="generated-url"` is generated. The contents of this field is the result of the URL recipe conversion. The urlRecipe value may refer to an existing metadata element by ${field[pattern/result/flags]}, which will take content of field and perform a regular expression conversion using the pattern given. For example: `${md-title[\s+/+/g]}` takes metadata element `title` and converts one or more spaces to a plus character.

zurl This is setting is mandatory and specifies the ZURL of the target in the form of host/database. The HTTP method should not be provided as this is guessed from the "sru" attribute value.

DATABASE parameters

Extra information may be carried in the Z39.50 Database or SRU path, such as authentication to be passed to backend etc. Some of the parameters override TARGET profile values. The format is

udb,parm1=value1&parm2=value2&...

Where udb is the unique database recognised by the backend and parm1, value1, .. are parameters to be passed. The following describes the supported parameters. Like form values in HTTP the parameters and

values are URL encoded. The separator, though, between udb and parameters is a comma rather than a question mark. What follows question mark are HTTP arguments (in this case SRU arguments).

The database parameters, in alphabetical order, are:

content-password The password to be used for content proxy session. If this parameter is not given, value of parameter `password` is passed to content proxy session.

content-proxy Specifies proxy to be used for content proxy session. If this parameter is not given, value of parameter `proxy` is passed to content proxy session.

content-user The user to be used for content proxy session. If this parameter is not given, value of parameter `user` is passed to content proxy session.

cproxysession Specifies the session ID for content proxy. This parameter is, generally, not used by anything but the content proxy itself when invoking Metaproxy via SRU.

nocproxy If this parameter is specified, content-proxying is disabled for the search.

password Specifies password to be passed to backend. It is also passed to content proxy session unless overriden by content-password. If this parameter is omitted, the password will be taken from TARGET profile setting authentication .

proxy Specifies one or more proxies for backend. If this parameter is omitted, the proxy will be taken from TARGET profile setting cfProxy. The parameter is a list of comma-separated host:port entries. Bost host and port must be given for each proxy.

realm Session realm to be used for this target, changed the resulting URL to be used for getting a target profile, by changing the value that gets substituted for the %realm string. This parameter is not allowed if access is controlled by auth_url in configuration.

retry Optional paramater. If the value is 0, retry on failure is disabled for the ZOOM module. Any other value enables retry on failure. If this parameter is omitted, then the value of `retryOnFailure` from the Torus record is used (same values).

torus_url Sets the URL to be used for Torus records fetch - overriding value of `url` attribute of element `torus` in zoom configuration. This parameter is not allowed if access is controlled by auth_url in configuration.

user Specifies user to be passed to backend. It is also passed to content proxy session unless overriden by content-user. If this parameter is omitted, the user will be taken from TARGET profile setting authentication .

x-parm All parameters that has prefix x, dash are passed verbatim to the backend.

SCHEMA

```
# Metaproxy XML config file schema

namespace mp = "http://indexdata.com/metaproxy"

filter_zoom =
  attribute type { "zoom" },
  attribute id { xsd:NCName }?,
  attribute name { xsd:NCName }?,
  element mp:torus {
    attribute allow_ip { xsd:string }?,
    attribute auth_url { xsd:string }?,
    attribute url { xsd:string }?,
    attribute content_url { xsd:string }?,
    attribute realm { xsd:string }?,
    attribute xsldir { xsd:string }?,
    attribute element_transform { xsd:string }?,
    attribute element_raw { xsd:string }?,
    attribute element_passthru { xsd:string }?,
    attribute proxy { xsd:string }?,
    attribute explain_xsl { xsd:string }?,
    attribute record_xsl { xsd:string }?,
    element mp:records {
      element mp:record {
        element mp:authentication { xsd:string }?,
        element mp:authenticationMode { xsd:string }?,
        element mp:piggyback { xsd:string }?,
        element mp:queryEncoding { xsd:string }?,
        element mp:udb { xsd:string },
        element mp:cclmap_au { xsd:string }?,
        element mp:cclmap_date { xsd:string }?,
        element mp:cclmap_isbn { xsd:string }?,
        element mp:cclmap_su { xsd:string }?,
        element mp:cclmap_term { xsd:string }?,
        element mp:cclmap_ti { xsd:string }?,
        element mp:contentAuthentication { xsd:string }?,
        element mp:elementSet { xsd:string }?,
        element mp:recordEncoding { xsd:string }?,
        element mp:requestSyntax { xsd:string }?,
        element mp:sru { xsd:string }?,
        element mp:sruVersion { xsd:string }?,
        element mp:transform { xsd:string }?,
        element mp:literalTransform { xsd:string }?,
        element mp:urlRecipe { xsd:string }?,
        element mp:zurl { xsd:string },
        element mp:cfAuth { xsd:string }?,
        element mp:cfProxy { xsd:string }?,
```

```
      element mp:cfSubDB { xsd:string }?,
      element mp:contentConnector { xsd:string }?,
      element mp:sortStrategy { xsd:string }?,
      element mp:sortmap_author { xsd:string }?,
      element mp:sortmap_date { xsd:string }?,
      element mp:sortmap_title { xsd:string }?,
      element mp:extraArgs { xsd:string }?,
      element mp:rpn2cql { xsd:string }?,
      element mp:retryOnFailure { xsd:string }?
    }*
  }?
}?,
element mp:fieldmap {
  attribute cql { xsd:string },
  attribute ccl { xsd:string }?
}*,
element mp:cclmap {
  element mp:qual {
    attribute name { xsd:string },
    element mp:attr {
      attribute type { xsd:string },
      attribute value { xsd:string }
    }+
  }*
}?,
element mp:contentProxy {
  attribute config_file { xsd:string }?,
  attribute server { xsd:string }?,
  attribute tmp_file { xsd:string }?
}?,
element mp:log {
  attribute apdu { xsd:boolean }?
}?,
element mp:zoom {
  attribute timeout { xsd:integer }?,
  attribute proxy_timeout { xsd:integer }?
}?
```

EXAMPLES

In example below Target definitions (Torus records) are fetched from a web service via a proxy. A CQL profile is configured which maps to a set of CCL fields ("no field", au, tu and su). Presumably the target

definitions fetched maps the CCL to their native RPN. A CCL "ocn" is mapped for all targets. Logging of APDUs are enabled, and a timeout is given.

```
 <filter type="zoom">
   <torus
      url="http://torus.indexdata.com/src/records/?query=%query"
proxy="localhost:3128"
      />
   <fieldmap cql="cql.anywhere"/>
   <fieldmap cql="cql.serverChoice"/>
   <fieldmap cql="dc.creator" ccl="au"/>
   <fieldmap cql="dc.title" ccl="ti"/>
   <fieldmap cql="dc.subject" ccl="su"/>

   <cclmap>
     <qual name="ocn">
       <attr type="u" value="12"/>
       <attr type="s" value="107"/>
     </qual>
   </cclmap>
   <log apdu="true"/>
   <zoom timeout="40"/>
 </filter>
```

Here is another example with two locally defined targets: A Solr target and a Z39.50 target.

```
   </filter>
   <filter type="zoom">
     <torus>
       <records>
         <record>
           <udb>ocs-test</udb>
           <cclmap_term>t=z</cclmap_term>
           <cclmap_ti>u=title t=z</cclmap_ti>
           <sru>solr</sru>
           <zurl>ocs-test.indexdata.com/solr/select</zurl>
         </record>
         <record>
           <udb>loc</udb>
           <cclmap_term>t=l,r</cclmap_term>
           <cclmap_ti>u=4 t=l,r</cclmap_ti>
           <zurl>lx2.loc.gov:210/LCDB_MARC8</zurl>
         </record>
       </records>
     </torus>
     <fieldmap cql="cql.serverChoice"/>
     <fieldmap cql="dc.title" ccl="ti"/>
   </filter>
```

SEE ALSO

metaproxy(1)

virt_db(3mp)

COPYRIGHT

Copyright (C) 2005-2015 Index Data

Appendix A

License

COPYRIGHT

Copyright (C) 2005-2015 Index Data

Metaproxy is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

Metaproxy is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Metaproxy; see the file LICENSE. If not, write to the Free Software Foundation, 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Appendix B

GNU General Public License

B.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

B.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

B.2.1 Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

B.2.2 Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

B.2.3 Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that

you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

B.2.4 Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

B.2.5 Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

B.2.6 Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

B.2.7 Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

B.2.8 Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

B.2.9 Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

B.2.10 Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

B.2.11 Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

B.2.12 NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

B.2.13 Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING

ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

B.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type “show w”. This is free software, and you are welcome to redistribute it under certain conditions; type “show c” for details.

The hypothetical commands “show w” and “show c” should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than “show w” and “show c”; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program “Gnomovision” (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.